Universidad Carlos III de Madrid

Escuela Politécnica Superior

Trabajo Fin de Grado

DESARROLLO DE UN SERVICIO DE POSICIONAMIENTO SOBRE UNA BASE DE DATOS DE INFORMACIÓN GEOGRÁFICA

José Luis González Martínez

Madrid – Septiembre 2016

Tutor: Jesús Arias Fisteus

Resumen

El servicio que se va a implementar en este proyecto es una biblioteca de código que pueda geo posicionar usuarios sobre la calle en la que se encuentren geográficamente. Además de proporcionar la posición geográfica de la calle en la que se encuentra el usuario, se tiene que proporcionar más datos referentes a la calle en la que se encuentra como son su nombre, superficie y velocidad máxima. También tiene que proporcionar datos referentes al usuario como son el sentido en el que esta recorriendo la calle y el porcentaje recorrido en ésta.

El principal objetivo de este proyecto es elaborar una biblioteca de código que proporcione una alternativa libre para obtener la posición geográfica de los usuarios. Se desarrolla en tecnologías como Python y Java para facilitar que otros desarrolladores puedan utilizar este servicio al elaborar sus aplicaciones. También tiene que ofrecer la posibilidad de implementarse mediante peticiones HTTP como alternativa a elaborar una base de datos local.

El presente documento tiene como finalidad la presentación del proceso completo llevado a cabo para el análisis, diseño e implementación del proyecto. Además se tratarán las principales tecnologías utilizadas, como son la base de datos geográfica y diversas tecnologías para implementar el proyecto. También se explicarán las metodologías de trabajo seguidas, junto a una planificación y presupuesto del proyecto. Por último, se incluyen una serie de pruebas de funcionamiento y las conclusiones obtenidas al realizar el proyecto

Abstract

The service that will be implemented in this project is a code library that allows users to know where they are geographically located. In addition to providing the geographical position of the street where the user is located, this project provides more data about the street where the user is located, such as the street name, surface and maximum speed. It also provides data related to the user like the direction of travel or the percentage of the street travelled.

The main objective of this project is to develop a code library that provides a free alternative for the user's geographical location. It is developed using technologies like Python and Java, in order to enable other developers, use this service in their applications. It must also offer the possibility to be implemented by HTTP requests as an alternative to implement a local database.

This whole document has the aim of describing the process taken to analyse, design and implement this project. Also there will be explained the main technologies used in this project, as the geographic database and diverse technologies used to implement it. In addition, it will be explained the working methodology followed in this project, together with a planning and a budget list of the project. Finally, there will be included some real case testing and the conclusions obtained from the completion of this project.

Índice

Lista de acrónimos

En esta sección se encuentran todos los acrónimos usados en este trabajo con el significado de éstos:

- GPS: Sistema de Posicionamiento Global.
- SIG: Sistema de Información Geográfica.
- BDG: Base de Datos Geográfica.
- OSM: OpenStreetMap.
- API: *Application Programming Interface.*
- HTTP: *Hypertext Transfer Protocol.*
- RAE: Real Academia Española.
- ETRS89: *European Terrestrial Reference System* 1989.
- NAD: *North American Datum.*
- WGS: *World Geodetic System.*
- ED50: *European Datum* 1950.
- GDD: Grados Decimales.
- REST: *REpresentational State Transfer.*
- WKT: *Welll-Known Text.*
- SRID: *Spatial Reference System Identifier.*
- URL: *Uniform Resource Locator.*
- GeoJSON: *Geographic JavaScript Object Notation.*
- CSV: *Comma-separated values.*
- KML: *Keyhole Markup Language.*
- HTML: *HyperText Markup Language.*
- ODbL: *Open Database License.*
- LOPD: Ley Orgánica de Protección de Datos.

Lista de figuras

1. Introduction

The following introduces the bachelor's thesis, exposing the motivation and objectives of the thesis, as well as the stages of it and the technologies that have been used along the development.

1.1. Motivation

Geographic information systems are on the order of the day. Proof of this is that all smartphones can know their location using the Global Positioning System (GPS) and also have a maps application. This coupled with other current trends like Big Data and analysis, interpretation and generalization of large amounts of information, are a great motivation for develop an application that translates geographic coordinates obtained by GPS in useful information about the user, in which the user's position is projected onto the road, the direction in which it circulates, the maximum speed and the surface in case that such information is available.

With this information obtained through multiple users, we can develop Geographic Information Systems (GIS), which are systems that transform this information with the aim of achieving results such as improving road safety or with a commercial purpose.

It's necessary to store this geographic information combined with user data in order to process it later. These databases are called geographic databases. With these databases coupled with logic to modify this information, can conform SIG systems.

Therefore, the geographic information generated in this work can serve as the basis for future applications that could transform all this geographic information

into useful information, which is sufficient motivation to do this work. The following will explain the objectives of this work.

1.2. Objectives

The main objective of this work is to implement a code library that allows to know the street where a vehicle is circulating and its projection on the street with high precision while minimizing errors.

Another objective is to develop this code library so it can be used in different programming languages, such as Java or Python. For this purpose, it must implement a web server and a web demonstrator, to test it and provide access to the code library.

Therefore, the ultimate objective to be achieved in this work is to develop a code library that allows to transform geographical information from multiple users to useful information than can be used for developing applications that can add value to this information. The following section is going to introduce the reader to the rest of the memory.

1.3. Introduction to the rest of the memory

In the following sections of this work the reader will be explained how the work is done, starting with the generic and ending in the most concrete.

First it will be explained the work plan followed with a briefly comment on each task and also the project budget.

Then the reader will be introduced in all the concepts and technologies used for the project, so that later the reader can understand easily the development and implementation of the project.

After introducing the reader about the technologies used, it will be described the project requirements for a proper operation. Also, it will be described the

architecture followed to meet these requirements, explaining in detail each component.

In subsequent sections it will described to the reader about the project design, the reasons why certain technologies are used including the algorithm design used and the implementation of the whole project, pointing out the most relevant aspects in the project implementation.

Finally, after finishing with project implementation, appropriate tests will be performed to verify that the requirements for simulated and real users are passed.

At the end, the regulatory framework will be analysed, describing the licenses necessary to do this work together with describing the laws related to managing data from real users. At least, it will be described the impact of this project will have on the economy and the society, focusing on improvements that can be achieved in applications that require the user geographical position.

1. Introducción

A continuación procedemos a introducir el trabajo de fin de grado exponiendo la motivación y objetivos del mismo e introducir los siguientes capítulos a tratar.

1.1. Motivación

Los sistemas de información geográfica están a la orden del día. Prueba de ello es que todos los *smartphones* pueden conocer su ubicación mediante el Sistema de Posicionamiento Global (GPS) y además cuentan con una aplicación de mapas. Esto juntado con otra tendencia actual como el Big Data y el análisis, interpretación y generalización de gran cantidad de información, suponen una gran motivación para realizar una aplicación que traduzca coordenadas geográficas obtenidas mediante el GPS en información útil respecto al usuario, en la que se encuentra la posición del usuario proyectada sobre la vía, el sentido en el que circula en ella, la velocidad máxima y la superficie en el caso de que se disponga esa información.

Con esta información obtenida a través de múltiples usuarios se pueden desarrollar Sistemas de Información Geográfica (SIG), los cuales son sistemas que transforman esta información con el objetivo de conseguir resultados como pueden ser mejorar la seguridad vial o con un fin comercial.

Esta información geográfica cruzada con datos respecto a los usuarios es necesario almacenarla para posteriormente poder operar con ella. A estas bases de datos se las denomina Bases de datos Geográficas (BDG). Con estas bases de datos sumada a cierta lógica para modificar la información se conforman los SIG.

Por tanto la información geográfica que se genera en este trabajo puede servir como base para futuras aplicaciones que puedan transformar toda esta información

geográfica en información útil, lo que supone una motivación suficiente para realizar este trabajo. A continuación se van a explicar los objetivos de este trabajo.

1.2. Objetivos

El principal objetivo de este trabajo es implementar una librería de código que permita obtener la calle por la que circula un vehículo y su proyección sobre ésta con alta precisión minimizando los errores.

Otro objetivo es desarrollar esta librería de código para que se pueda utilizar en diferentes lenguajes de programación, como pueden ser Java o Python. Para ello se tiene que implementar un servidor web y un demostrador web para poder realizar pruebas y facilitar el acceso a la librería de código.

Por tanto el objetivo final que se quiere conseguir en este trabajo es elaborar una librería de código que permita transformar información geográfica de múltiples usuarios para que posteriormente esta información geográfica se pueda emplear para realizar aplicaciones que puedan aportar un valor añadido. En el siguiente apartado se va a introducir al lector al resto de la memoria.

1.3. Introducción al resto de la memoria

En las siguientes secciones del trabajo se explicará al lector como se ha realizado el trabajo, empezando en lo más genérico y terminando en lo más concreto.

Primero se explicará el plan de trabajo seguido para la realización del trabajo comentando brevemente cada tarea a realizar y el presupuesto necesario para cumplimentar este proyecto.

A continuación se introducirá al lector en todos los conceptos y tecnologías utilizados para realizar el proyecto, para que posteriormente el lector pueda comprender con mayor facilidad el desarrollo e implementación del proyecto.

Después de introducir al lector sobre las tecnologías empleadas para la realización del proyecto se comentan los requisitos que éste debe cumplir para un

correcto funcionamiento y la arquitectura seguida para cumplir estos requisitos, explicando detalladamente cada componente por separado.

En las secciones posteriores se comentara al lector el diseño del proyecto, en el cual se determinan las razones por la cual se utilizan ciertas tecnologías y el diseño del algoritmo utilizado y la implementación de todo el proyecto comentando los aspectos mas relevantes.

Por último, después de acabar con la implementación del proyecto se realizaran las pruebas pertinentes para comprobar que se cumplen los requisitos establecidos tanto para recorridos simulados como para recorridos de usuarios reales.

Finalmente se analizará el marco regulador, comentando las licencias necesarios para realizar este trabajo y leyes contempladas en el uso de datos provenientes de usuarios reales y para acabar se analizará el impacto que tendrá este proyecto tanto en la economía como en la sociedad, centrándose en las mejoras que se pueden conseguir en aplicaciones que precisen de utilizar la posición geográfica del usuario el cual las usa.

2. Plan de trabajo y presupuesto

En esta sección se explicará el plan de trabajo utilizado para cumplir con los objetivos citados anteriormente y el presupuesto para realizarlos.

2.1. Plan de trabajo

Para llevar a cabo el proyecto se proponen las siguientes actividades:

1. Estudio de los sistemas SIG: Conocer qué es un sistema SIG y sus componentes.
    a. Tipos de coordenadas.
    b. Bases de datos geográficas.
    c. Funcionalidades del GPS.
2. Elección de base de datos geográfica: Se procede a elegir y familiarizarse con una base de datos geográfica.
    a. Se decide entre OpenStreetMap (OSM) y Google Maps.
    b. Modelo de datos geográfico de OSM : Conocer las tablas con sus campos y relaciones que conforman la base de datos geográfica.
    c. Herramientas para importar a una base de datos geográfica con los datos provenientes de OSM a local: Conocer sobre qué tipo de base de datos se deben importar los datos y las herramientas disponibles para esta tarea.
3. Importación de datos geográficos de España a una base de datos geográfica local.
    a. Se documenta sobre la base de datos PostgreSQL y posteriormente se instala.

b. Uso de la herramienta de importación osm2pgsql.

4. Implementación de consultas a la base de datos geográfica: Se construyen las consultas a la base de datos geográfica para lograr el objetivo de este trabajo.
    a. Métodos Postgis: Estudio de la *Application Programming Interface* (API) de Postgis.
    b. Se crean las consultas necesarias.
        i. *Reverse geocoding*.
        ii. Proyectar un punto GPS sobre una calle.
        iii. Obtener toda la información necesaria de una calle.
        iv. Validar todas las consultas.

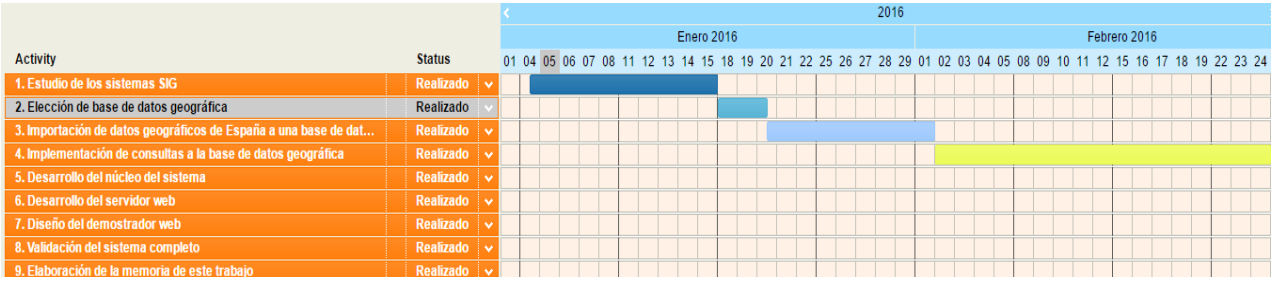

Figura 2.1 Diagrama de Gantt de las actividades del 1-4

5. Desarrollo del núcleo del sistema: Se desarrolla el bloque principal de la aplicación.
    a. Se elige el lenguaje en el que se basará la aplicación.
    b. Se documenta sobre Python.
    c. Conexión a la base de datos geográficas: Se implementa la conexión a la base de datos geográfica.
    d. Desarrollar e implementar el algoritmo principal de este trabajo.
6. Desarrollo del servidor web.
    a. Se decide el *framework* para la implementar el servidor web.
    b. Se documenta sobre Flask.
    c. Desarrollo e implementación del servidor web para sr accesible mediante peticiones *Hypertext Transfer Protocol* (HTTP).
7. Diseño del demostrador web.
    a. Se documenta sobre Leaflet.
    b. Se desarrolla e implementa la interfaz web.

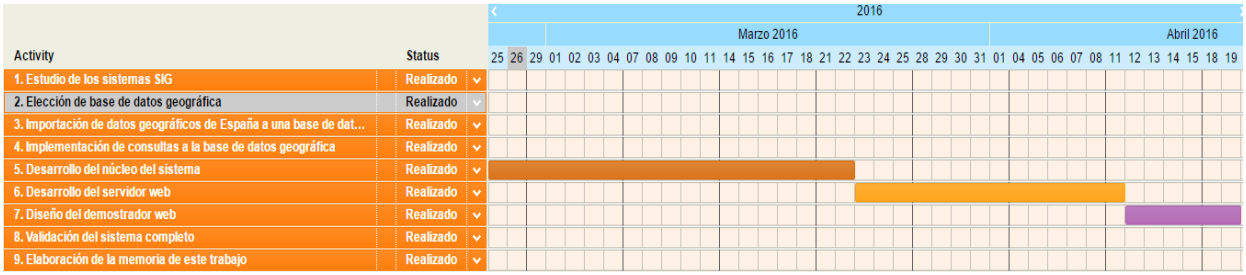

Figura 2.2 Diagrama de Gantt de las actividades del 5-7

8. Validación del sistema completo: Se realizan las pruebas necesarias para validar la aplicación.
    a. Pruebas unitarias.
    b. Pruebas con recorridos reales.
9. Elaboración de la memoria de este trabajo.

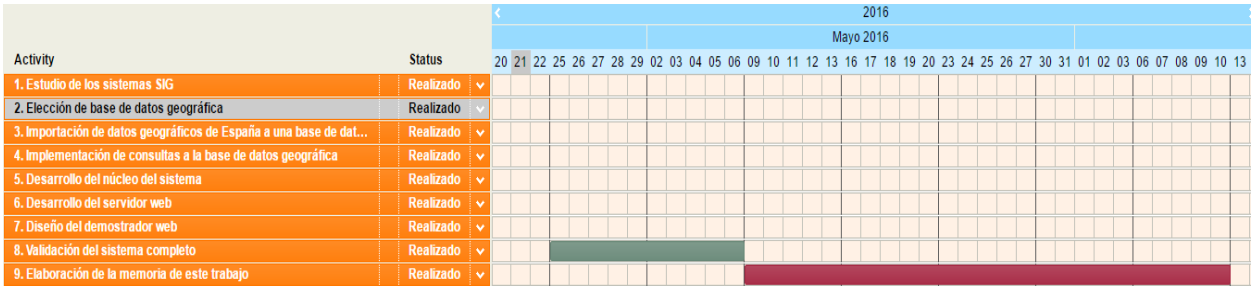

Figura 2.3 Diagrama de Gantt de las actividades del 8-9

A continuación después de comentar las actividades del proyecto y su duración se procede a exponer el presupuesto del trabajo.

2.2. Presupuesto

Para la realización de este trabajo se necesitan los siguientes recursos humanos y materiales, los cuales se exponen en el siguiente presupuesto:

- Materiales: En este apartado del presupuesto se incluye el precio de los materiales empleado restando la amortización, que en este caso será del 26% al tratarse de equipo informático.
    - Portátil Lenovo G50-80, i3-5005U, 8GB RAM y 1TB: 534 € - 138 € = 396 €
- Recursos humanos: En este apartado se incluye el salario de los componentes que forman este trabajo durante un año.
    - Programador junior: 27.000 €
    - Jefe de proyecto: 6.700 €
- Presupuesto final: Sumando los dos presupuestos anteriores se obtiene el coste total para realizar este trabajo.
    - Total: 34.096 €

Después de calcular el presupuesto total de proyecto se va a introducir al lector más en profundidad en las tecnologías empleadas en este trabajo.

3. Estado del arte (antecedentes)

La evolución de los SIG en estos últimos años ha sido notablemente alta, en este apartado se intenta poner en contexto al lector sobre el estado actual y las principales tecnologías de estos sistemas.

3.1. Sistema de Información Geográfica (SIG)

Vivimos en un mundo en el que la información georreferenciada, esto es, asociada a posiciones geográficas, es cada vez mayor. Por esa razón, la cartografía ha pasado de tener un papel auxiliar en otras tecnologías a tener un papel cada vez más importante, no solo en el ámbito científico y tecnológico, sino también en el ámbito social.

La cartografía ha sufrido un increíble desarrollo en las últimas décadas, en este tiempo se han producido, entre otras cosas la digitalización de los mapas y la creaciones de algoritmos cada vez más eficientes para generar la ruta más corta entre dos puntos. Esto ha sido posible gracias al desarrollo de herramientas SIG, que han provocado además del desarrollo tecnológico antes citado, un desarrollo social, aumentando la calidad de vida de gran parte de la población.

Prueba de este desarrollo sería comparar una situación cotidiana como puede ser ir a una dirección que se ha proporcionado, hace 20 años y ahora. Antes se tenía que buscar en un callejero o mapa la calle indicada y buscar la mejor manera de ir. Sin embargo, ahora basta con introducir la dirección en nuestro dispositivo móvil.

Después de esta breve introducción de los SIG, su importancia y evolución, vamos a definir en qué consiste un SIG. Un SIG [1] como mínimo puede almacenar, interpretar, y analizar datos geográficos para conseguir un objetivo.

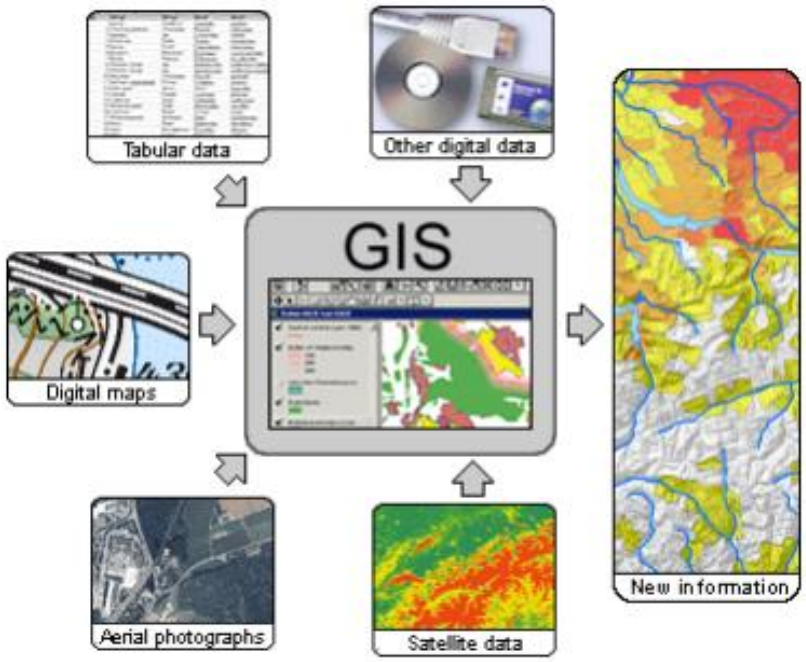

Figura 3.1 Componentes de un SIG [1]

Un SIG es un sistema informático, incluyendo software, hardware, datos y aplicación para conseguir un objetivo.

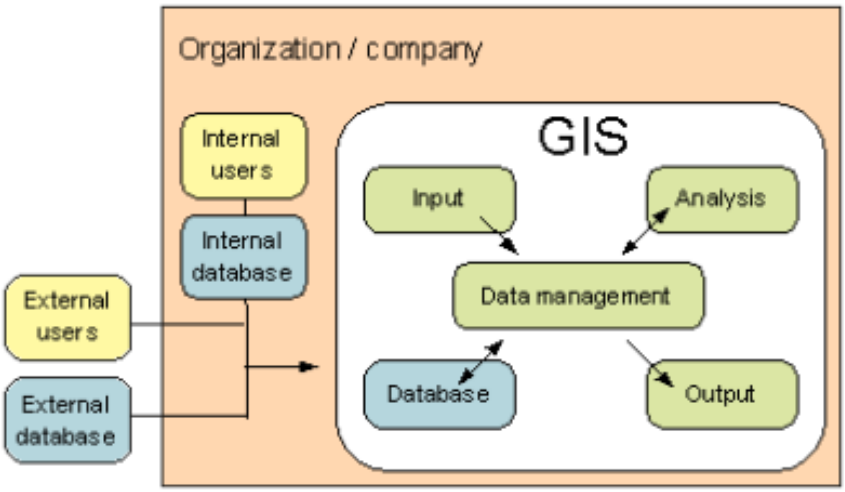

Figura 3.2 Estructura de un SIG [1]

Después de introducir al lector sobre los SIG, en las siguientes secciones se van a introducir los conceptos y tecnologías utilizadas para recolectar y almacenar estos geográficos.

El primer concepto fundamental en un SIG es el sistema de coordenadas, ya que gracias a se puede asignar una posición geográfica a un objeto.

1. Sistemas de coordenadas geográficas

A la hora de crear un sistema de información geográfica (SIG), es de vital importancia conocer cómo ha llegado el ser humano a establecer las referencias que sirvan de base para luego poder posicionar elementos sobre la superficie terrestre.

Las coordenadas geográficas [2] son un sistema universal para la localización de puntos sobre la superficie terrestre. Este sistema se basa en un conjunto de anillos o círculos imaginarios que rodean a la esfera terrestre. Tal como se indica en la figura 3.3:

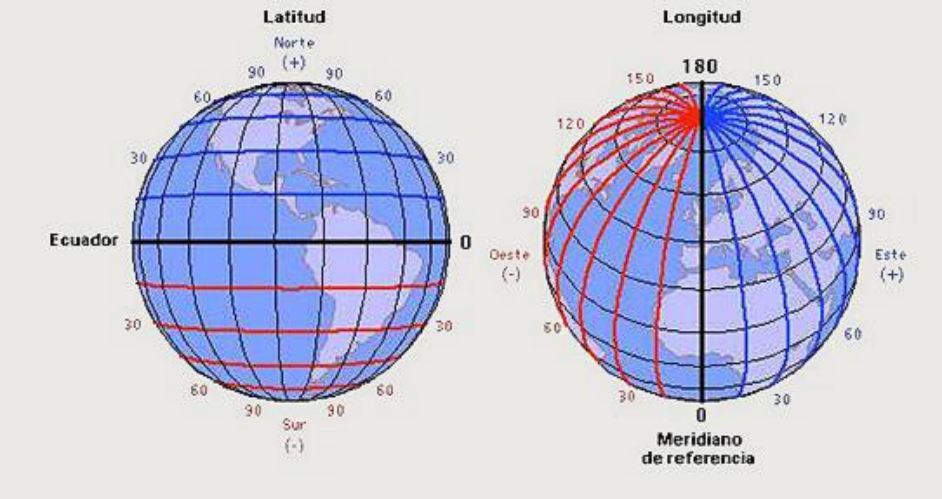

Figura 3.3 Paralelos y meridianos terrestres [2]

Gracias a estos círculos o anillos se puede localizar cualquier punto de la superficie terrestre. Se encuentran dos tipos de círculos.

1. Paralelo: Tal como define la Real Academia Española (RAE), "Cada uno de los círculos menores paralelos al ecuador, que se suponen descritos en el

globo terráqueo y que sirven para determinar la latitud de cualquiera de sus puntos o lugares". Cabe aclarar que el ecuador es el plano perpendicular al eje de rotación de la Tierra y que pasa pos su centro, dividiendo la superficie terrestre en dos mitades o hemisferios.

2. Meridiano: Como indica la RAE, se puede definir un meridiano como: "Cada uno de los semicírculos de la esfera terrestre que van de polo apolo."

   A contrario que los paralelos, sirven para delimitar la longitud de un punto o lugar, desde el meridiano de Greenwich, también conocido como meridiano cero, ya que se corresponde a la circunferencia imaginaria que une los polos terrestres.

Por tanto para localizar un punto basta con indicar su latitud y su longitud, estas medidas se expresan en unidades de medida angular, en grados. Variando la latitud de -90º a 90º, siendo las latitudes positivas las correspondientes al hemisferio norte y las negativas las correspondientes al hemisferio sur. Al contrario de la latitud, la longitud varía de -180º a 180º, siendo las longitudes positivas las correspondientes a los puntos al este del meridiano de Greenwich, y las negativas las encontradas al oeste de este meridiano.

Sin embargo, partiendo de la base de que la Tierra no forma una esfera perfecta sino que cada región tiene un relieve o una distancia al centro de la tierra determinado, asemejándose en mayor medida a un elipsoide, cada país o región se vio en la necesidad de tomar un sistema de referencia específico. Por ello antes de continuar con la explicación es conveniente introducir al lector sobre dos conceptos previos:

1. El geoide: Definido como todos los puntos de la superficie terrestre de igual gravedad y sin variables externas como las mareas o la atracción de la luna.

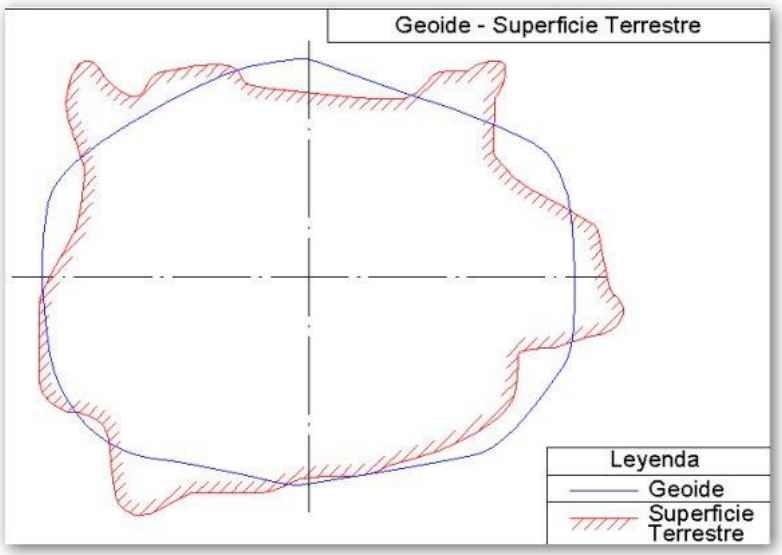

Figura 3.4 Representación del geoide [3]

2. El elipsoide: Definido como la figura geométrica que más se asemeja a la forma de la tierra a cartografiar. Dicha figura geométrica se define matemáticamente por el radio mayor (a), el radio menor (b) y el aplastamiento del elipsoide (1/f = 1-(b/a))

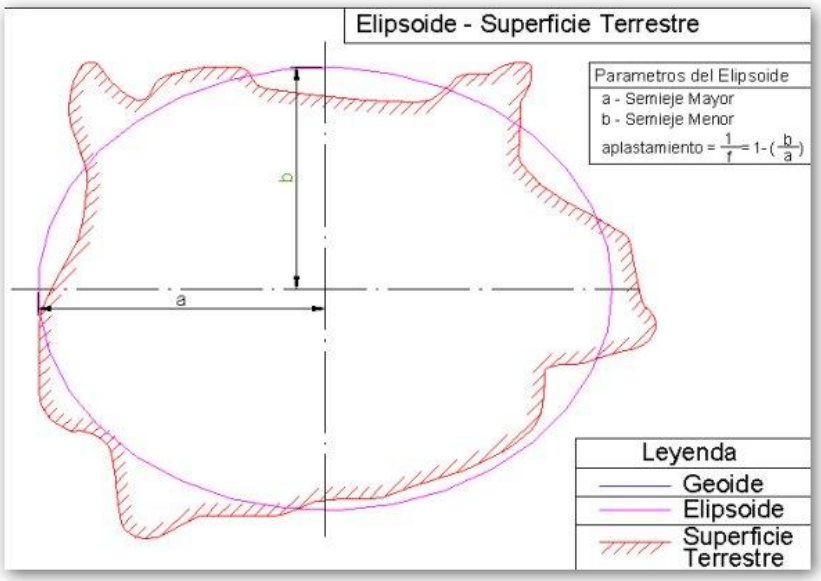

Figura 3.5 Representación del elipsoide [3]

Una vez definidos ambos conceptos previos, se puede definir el datum [3] como el punto tangente al elipsoide y al geoide donde ambos son coincidentes. Dicho datum es definido por el elipsoide (compuesto por los radios (a y b) y su aplastamiento) y un punto llamado fundamental donde el elipsoide y la tierra son tangentes.

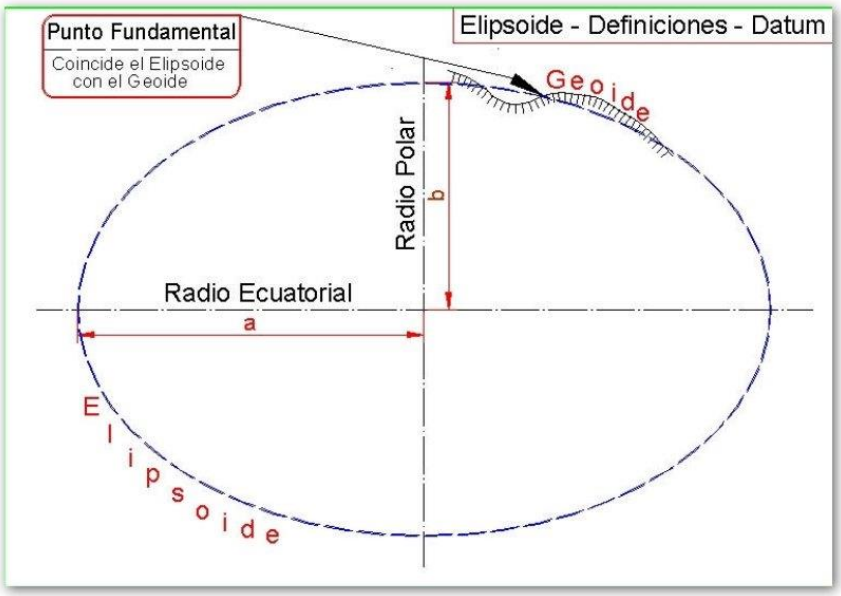

Figura 3.6 Representación del datum [3]

Por lo explicado anteriormente surgen infinidad de datum repartidos por todo el mundo. Por ejemplo, como datum específico para una región se encuentran:

- *European Terrestrial Reference System* 1989 (ETRS89) es el datum oficial en España que entró en vigor en 2015.
- *North American Datum* (NAD) es el datum usado como sistema de referencia para norte américa. NAD define dos elipsoides diferentes NAD27 definido por el elipsoide de Clarke de 1866 y NAD83 definido por el elipsoide GRS 80.

Estos datum son específicos para esas regiones y no pueden ser usados en otro territorio. Sin embargo existe el *World Geodetic System* (WGS), que es un estándar a nivel mundial que cuenta con varias versiones, el WGS72 creado por el

departamento de defensa de Estados unidos a principios de los años 70, y el WGS84 que es una versión más actualizada.

Actualmente en España, según se indica en el BOE núm. 207 del miércoles 29 de agosto de 2007 [4], a partir del 1 de Enero de 2015 se estableció como sistema oficial en España el ETRS89 , permitiendo así una completa integración de la cartografía oficial en el país . Ya que el territorio de las islas Canarias no soportaba *European Datum* 1950 (ED50) y usaba un sistema propio, REGCAN95.

Después de explicar al lector como se localiza un punto en la superficie terrestre, se procede a introducir al lector que tecnología se usa para obtener estas coordenadas citadas anteriormente.

2. Sistema de Posicionamiento Global (GPS)

El Sistema de Posicionamiento Global o GPS [5], es un sistema global de navegación por satélite que permite determinar en cualquier lugar de la Tierra la posición de un objeto, una persona o un vehículo, con gran precisión las 24 horas del día. Funciona mediante una red de 27 satélites (24 operativos y 3 de respaldo) en órbita a 20.200 km sobre el globo terráqueo, con trayectorias sincronizadas para cubrir toda la superficie de la Tierra.

La idea sobre su funcionamiento es bastante simple, ya que al poder determinar la distancia de un punto de la superficie a tres satélites distintos se puede triangular la posición del punto sin dificultades. Para calcular dichas distancias los satélites GPS envían continuamente señales radio compuestas por dos canales, dos códigos y un mensaje de navegación. Por tanto cuando un receptor quiere averiguar su posición GPS, simplemente recoge estas señales mediante su antena y gracias a su software incluido calcula esas tres distancias, y a partir de ellas la posición GPS, tal como se indica en la figura 3.7. Sin embargo también es necesario un cuarto satélite para otros cálculos.

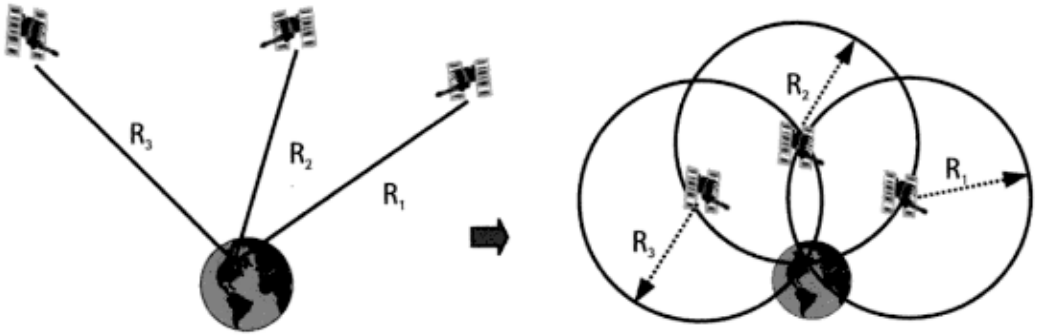

Figura 3.7 Calculo de posición GPS mediante tres satélites [5]

En cuanto a la precisión que se obtiene con dicho método es de 100 m horizontales 156 verticales y 340ns de tiempo. Para mejorarla se emplea el método diferencial al calcular la distancia a los satélites, en el cual se emplean dos receptores de señal simultáneamente. El sistema GPS también permite ser usado para otros fines como puede ser calcular la velocidad de un objeto.

A continuación después de conocer como se representan los puntos en la superficie terrestre y la tecnología utilizada para obtener dicha posición. Solo queda por explicar dónde se almacenan dichas posiciones junto con información para conformar un SIG.

3. Base de Datos Geográfica (BDG)

Se puede definir una BDG [6] como un conjunto de datos geográficos organizados de tal forma que permiten su interpretación y análisis dentro de una aplicación SIG.

Estos datos siguen un modelo de datos, que es la parte más importante de la BDG, que indicarán cómo se almacena un elemento geográfico dentro de la BDG y se relaciona con otros objetos. Este modelo depende del proveedor del cuál se hayan extraído los datos, como veremos más detalladamente en el apartado de diseño de este trabajo.

Después de comentar ciertos aspectos importantes sobre una aplicación SIG, a continuación se presentan los diferentes proveedores de datos geográficos para

elaborar la BDG, y los posibles gestores de bases de datos para almacenar esos datos.

3.2. Proveedores de datos geográficos.

En cuanto los proveedores de mapas, se encuentran dos opciones para proporcionar los datos geográficos necesarios para desarrollar una aplicación, bien sean en su totalidad como OSM o mediante una API como Google Maps. Por las razones explicadas en el apartado de diseño (referencia), se escoge OSM como proveedor de datos geográficos.

3.3. OpenStreetMaps (OSM)

Tal como se indica en su documentación [7] : "OpenStreetMap (también referido como OSM) es un proyecto colaborativo para crear un mapa libre y actualizable de todo el mundo; por medio de una comunidad de usuarios, es decir personas con un objetivo común, que ceden su tiempo desinteresadamente y sin fines de lucro para tener la posibilidad de ver, copiar, modificar, y usar información geográfica (como esta) de cualquier parte del mundo sin restricciones de ningún tipo."

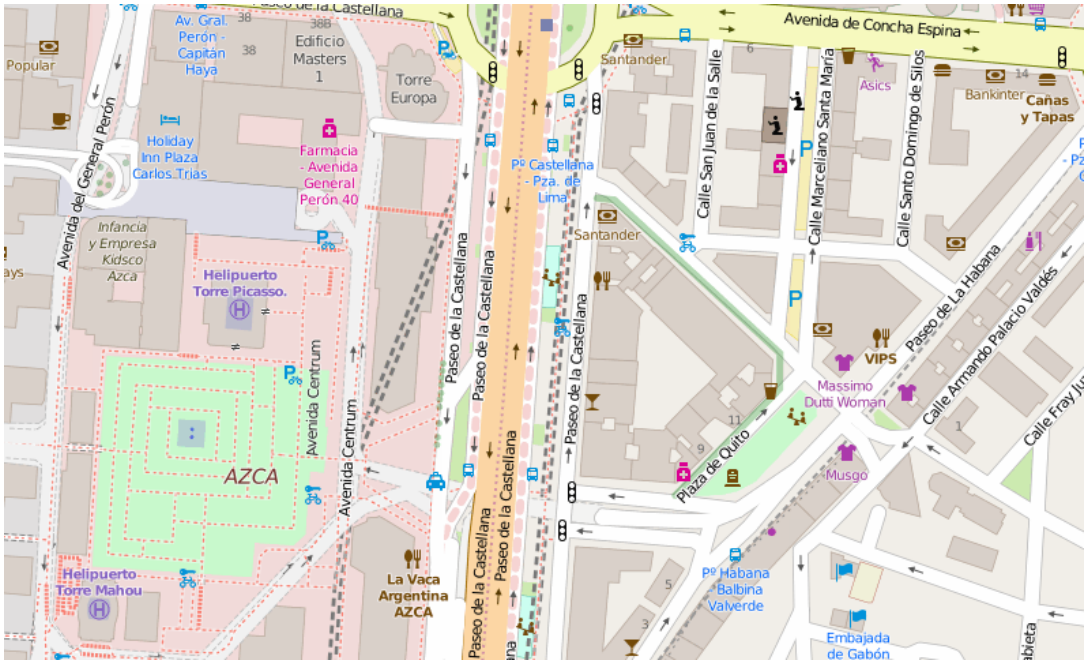

Figura 3.8 Mapa de OSM de Madrid [8]

Por tanto OSM trata de crear un mapa colaborativo de la misma forma que Wikipedia ha creado una enciclopedia. Al tratarse de un proyecto colaborativo, tu puedes visualizar y editar los datos, para esas funciones OSM consta principalmente de tres herramientas:

- Un visor: Para ver y consultar detalles del mapa.
- Editores online: A través de un *applet* Java y sin necesitar más que de un navegador web es posible introducir nuevos datos como calles, puntos de interés, zonas, etc. o etiquetar los ya existentes. Para ello es necesario registrarse, y el sistema guarda un historial de cambios y sus autores.
- Editores offline: Son programas que se pueden instalar en el ordenador del usuario. Poseen herramientas para facilitar la tarea de digitalización y etiquetado de nueva información.

Pero el aspecto más importante de un proveedor de mapas para este trabajo, es el modelo de datos que sigue, ya que se necesitará analizar y trabajar con el.

1. Modelo de datos de OSM

El modelo de datos que emplea OSM [7], está compuesto por tres principales tipos de estructuras:

- Nodo: Es la estructura más simple en OSM, ya que representa un punto en el espacio, definido por los siguientes parámetros: id, latitud, longitud y opcionalmente una etiqueta, para indicar alguna propiedad adicional correspondiente al punto. Un nodo también puede formar parte de una vía.

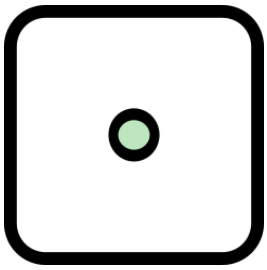

Figura 3.9 Representación de un punto en OSM

- Vía: Es una lista de 2 a 2000 nodos. Se utiliza para representar cualquier tipo de vías o polígonos, como por ejemplo un edificio o una ciudad. Está definido como mínimo por los siguientes parámetros: id y geometría. En este último se indica la lista de nodos que componen la vía o polígono.

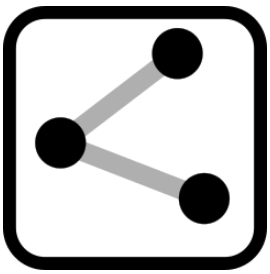

Figura 3.10 Representación de una línea en OSM

- Relaciones: Sirven para relacionar nodos y vías. Se pueden usar tanto para relacionar varias vías que forman una ruta como para indicar si una vía es accesible desde otra con la cual intersecciona.

Figura 3.11 Representación de una relación en OSM

Ahora que se conoce OSM como proveedor de datos geográfico y su modelo de datos, es conveniente conocer de qué herramientas se dispone para extraer los datos de estos mapas proporcionados por OSM para posteriormente importarlos a una BDG local.

2. Herramientas de exportación

Para la extracción de los datos geográficos de OSM se dispone de dos principales métodos, dependiendo del tamaño de la región que se necesite exportar:

- Para pequeñas regiones: Se dispone de una herramienta dentro de OSM que permite exportar los datos de una pequeña región del mapa.

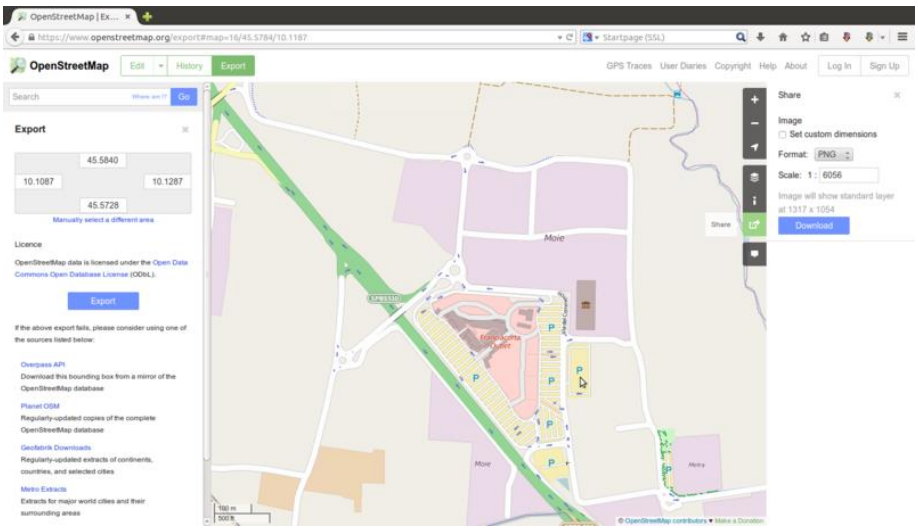

Figura 3.12 Captura de como exportar datos de OSM [8]

También se dispone de esta funcionalidad mediante una llamada HTTP, en la cual se indican los límites de la región en el siguiente orden: Mínima longitud, mínima latitud, máxima longitud, máxima latitud. Todos estos datos se expresan en grados decimales (GDD), teniendo en cuenta que tanto la latitud sur como longitud oeste son negativas. Un ejemplo de cómo usar esta funcionalidad sería esta petición ejemplo:

"http://api.openstreetmap.org/api/0.6/map?bbox=11.54,48.14,11.54,48.14"

- Para grandes regiones: Para grandes cantidades de datos, o incluso el mapa completo de OSM, se dispone de sitios web de terceros que proporcionan los datos geográficos de OSM. En la documentación de OSM se indica la frecuencia de mantenimiento de los datos geográficos y las regiones que se pueden exportar desde cada uno de ellos. En el apartado de diseño. se expondrá y justificará la elección de uno de estos sitios web.

El siguiente paso después de obtener los datos geográficos de OSM, es cómo importar estos datos a una BDG. Para ello se emplea la herramienta Osm2pgsql.

3. Osm2pgsql

Osm2pgsql es una herramienta que permite importar los datos geográficos procedentes de OSM a PostgreSQL, con posibilidad de habilitar PostGIS. Ambas tecnologías se explicarán más adelante este apartado.

En cuanto a sus características permite elegir la proyección para los datos geográficos importados y acepta varios formatos para los datos de entrada (.gz, .bz2, .pbf y .o5m). Por otra parte, está disponible en todos los sistemas operativos, lo que facilita su uso.

Los datos geográficos importados procedentes de OSM siguen el siguiente esquema dentro de la BDG. Almacenándose en dos tipos de tablas:

- Tablas de datos procesados: Se importan los datos procedentes de OSM. Contienen como mínimo dos índices:

- Osm_id: Id que identifica el punto, línea o polígono.
- Way: Geometría del objeto, las coordenadas que definen el punto, línea o polígono.

Dentro de este tipo de tablas se encuentran cuatro tablas, cada una cumple una función diferente.

- Planet_osm_line: Almacena todas las líneas importadas.
- Planet_osm_point: Almacena todos los nodos que posean al menos una etiqueta, los nodos empleados para dibujar una calle no son importados.
- Planet_osm_polygon: Almacena todos polígonos y líneas cerradas.
- Planet_osm_roads: Almacena un subconjunto de las vías almacenadas la tabla planet_osm_line. Esta tabla se emplea para renderizar el mapa.

- Tablas intermedias: Estas tablas solo existen si se elige el modo *slim* en la importación. Únicamente contienen los datos antes de introducirlos en las tablas de datos procesados. Dentro de este tipo se encuentran estas tablas: planet_osm_nodes, planet_osm_ways y planet_osm_rels.

Para cerrar este apartado, después de tratar el proveedor de datos geográficos elegido, comentar su modelo de datos y las herramientas que se disponen para exportar estos datos y posteriormente importarlos a una BDG. Únicamente queda por explicar con que tecnología se pueden representar estos datos en un navegador web o móvil.

4. Leaflet

Leaflet [9] es una librería de código abierto de JavaScript, que permite tanto renderizar mapas como interaccionar con ellos. Leaflet está diseñado para funcionar de manera eficiente en todas las principales plataformas de escritorio y móviles, y también se puede ampliar con una gran cantidad de *plugins*, con una bien documentada API y un código simple en el cual es fácil contribuir.

Figura 3.13 Logotipo de Leaflet [9]

Después de conocer más a fondo el proveedor de mapas OSM y todas las herramientas ofrecidas para exportar, importar y representar los datos geográficos de OSM se continuará con las tecnologías usadas para almacenar la información y conformar las BDG.

3.4. Bases de datos

Antes de entrar en detalle con la tecnología que se usará como gestor de bases de datos para almacenar los datos geográficos de OSM, se comentarán brevemente las características de una base de datos objeto-relacional.

1. Base de datos objeto-relacional

Una base de datos relacional es una base de datos que sigue con el modelo relacional, en el que sus elementos poseen relaciones, lo que facilita su interpretación por el usuario. Un ejemplo de este tipo de base de datos sería PostgreSQL.

Este tipo de base de datos encaja perfectamente con el modelo de datos ofrecido por OSM, por lo que elegiremos este tipo de base de datos. Concretamente se elige PostgreSQL como sistema de gestión de bases de datos para este trabajo.

2. PostgreSQL

PostgreSQL [10] es un potente sistema de gestión de bases de datos objeto-relacional, que se distribuye bajo licencia de código abierto BSD, con 20 años con desarrollo activo como código libre, y con 30 años desde su creación. En los que

tanto el número de desarrolladores como de líneas de código ha ido aumentando año a año, tal como se puede observar en la figura 3.14.

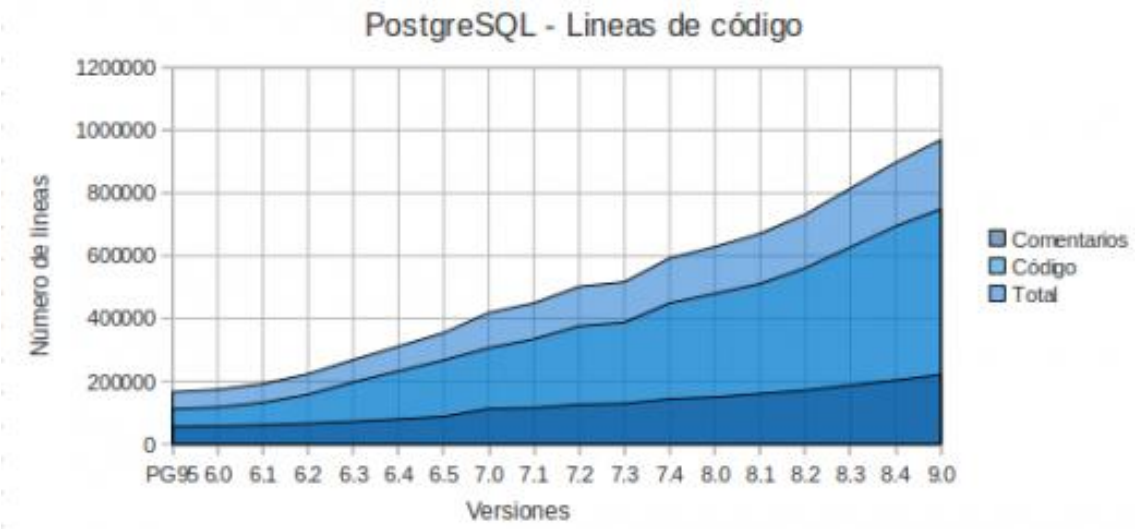

Figura 3.14 Gráfica con el incremento de líneas de código en PostgreSQL [10]

PostgreSQL funciona para la mayoría de sistemas operativos incluyendo Linux, otros sistemas UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows. Soporta claves externas, *joins views* y *triggers*. Incluye tipos de datos SQL: 2008 como INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL y TIMESTAMP. También posee interfaces para programar en C/C++, Java, .Net, Perl, Python y Ruby entre otros.

Utiliza un modelo cliente/servidor usando multiprocesos para garantizar la estabilidad del sistema. Esto implica que un fallo en uno de los procesos no afectará al sistema y continuará funcionando. En cuanto a su estructura contiene los siguientes componentes mostradas en la figura 3.15

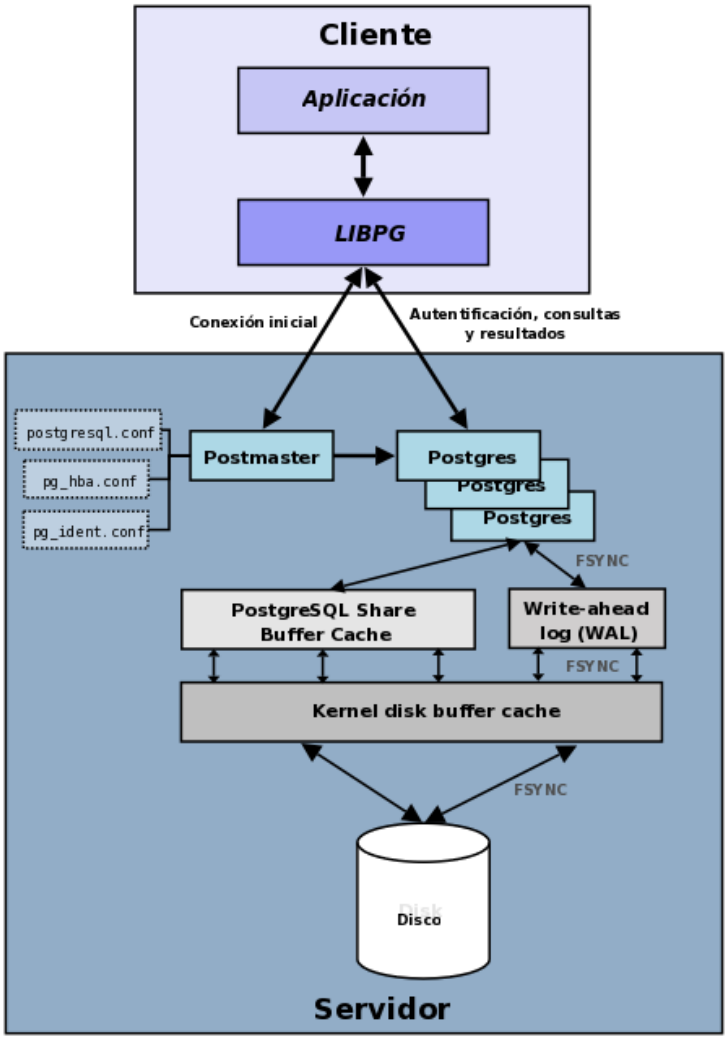

Figura 3.15 Estructura de PostgreSQL

También dispone de software como PGAdmin III que permite al desarrollador administrar, gestionar y construir consultas con mayor facilidad. Y permite habilitar PostGIS, una tecnología muy importante para operar con objetos geográficos.

3. PostGIS

PostGIS [11] es una extensión de PostgreSQL que ofrece funcionalidades extra a la base de datos. Proporciona numerosos métodos para calcular distancias,

proyecciones y transformaciones de coordenadas entre otros. También optimiza las consultas a la base de datos, convirtiéndolas más rápidas y eficientes.

Con las tecnologías explicadas anteriormente se puede elaborar una BDG con datos geográficos de OSM. Por lo que únicamente queda por explicar la tecnología utilizada para implementar la aplicación sobre la BDG. Tal como se indica en la sección de diseño se optó por desarrollar la aplicación en Python.

3.5. Python

Python es un claro y potente lenguaje de programación orientado a objetos comparable con Perl, Ruby o Java. Se puede considerar un lenguaje moderno, fue inventado en 1991 por Guido van Rossum, que consiguió un lenguaje de programación realmente productivo.

En sus comienzos Python se utilizaba como alternativa a Perl entre los usuarios de UNIX, hasta que entre 2003 y 2007 llegó su mayor expansión con el auge de las aplicaciones web y la nube. De hecho Python es uno de los cuatro lenguajes oficiales de Google y toda la infraestructura de Youtube está programada en Python. Además Python es el lenguaje que utilizan la mayoría de *startups*. Sus características son:

- Usa una sintaxis clara, que implica que los programas sean fáciles de leer y entender.
- Es un lenguaje fácil de usar. Permite conseguir objetivos con facilidad.
- Cuenta con una amplia librería de código, que permite acciones, como conectar a un *webserver* o usar expresiones regulares, sin dificultad.
- Posee un modo interactivo, ideal para programar pequeños fragmentos de código. Además para proyectos de mayor tamaño cuenta con entornos de desarrollo como IDLE o Eclipse aplicando una extensión..
- Es un lenguaje modular, ya que es fácil añadir nuevos módulos programados en un lenguaje compilado como C o C++, aumentando así el rendimiento.
- Funciona en la mayoría de sistemas operativos como Windows, MacOS y la mayoría de marcas de UNIX.
- Es un lenguaje libre en todos los sentidos ya que no implica ningún coste descargar o usar Python y porque puede ser modificado y redistribuido

debido a su licencia de código abierto. Gran parte del éxito de Python se debe a la gran cantidad de documentación que existe sobre el lenguaje como en sus librerías.

Concretando para este trabajo, se necesita un *framework* capaz de implementar Python en una aplicación REST (*REpresentational State Transfer*), para ello se cuente con *frameworks* como Flask y Django, los cuales compararemos en la sección de diseño del trabajo.

Terminada la introducción al lector sobre los conceptos y tecnologías usadas para realizar este trabajo. A continuación se comentarán los requisitos que debe cumplir la aplicación.

4. Requisitos

Una vez presentados los objetivos de la aplicación y las tecnologías utilizadas, en esta sección se describe de modo general la funcionalidad que el sistema debe proporcionar. Para ello, antes de comentar los requisitos de este proyecto, se necesitara un entorno de trabajo con las siguientes características:

- BDG de España partiendo de datos geográficos de OSM, mediante el uso de una herramienta de importación.
- Conexión con la BDG de España desde la aplicación.
- Datos reales de recorridos para validar correctamente el trabajo.

Después de comentar las características del entorno de trabajo sobre el que se desarrollará este proyecto. Se van a enumerar y explicar los requisitos finales de este trabajo:

- Tiene que ofrecer la posibilidad de ser integrado en otras aplicaciones desarrolladas en Java o Python.
- Al recibir un punto o un recorrido se deberán devolver los siguientes datos:
    - Calle en la que se encuentra el punto GPS: Se decidirá con el menor error posible la calle o carretera en la que se encuentra el usuario que ha enviado la posición GPS y se devolverá el identificador de la calle o ruta.
    - Punto proyectado sobre la calle o carretera en la que se encuentra el usuario.
    - Porcentaje recorrido: Se devolverá también el porcentaje recorrido por parte del usuario de la calle en la que se encuentre situado.
    - Sentido: También se indicará si el usuario está recorriendo la calle del principio al final de la misma o viceversa.

- Al recibir un identificador de una calle o ruta se debe devolver la siguiente información:
    - Nombre de la calle o ruta.
    - Velocidad máxima, en el caso de que conste en la base de datos.
    - Superficie de la calle o ruta, en el caso de que se disponga de esta información.
    - Cualquier otro atributo que pueda ser necesario, y que se encuentre entre los datos proporcionados por OSM.
- Se debe poder acceder al servicio mediante peticiones HTTP, por lo tanto se debe implementar un servidor web.
- Usando el servidor web, se debe implementar un demostrador web, que cumpla con los siguientes requisitos:
    - Representación del mapa en el demostrador web.
    - Interacción con el mapa para representar puntos o recorridos pasando por la aplicación.
    - Carga de recorridos para validar el trabajo con recorridos reales. Estos recorridos tienen que poder ser representados en el mapa pasando por la aplicación y sin pasar por ella para realizar análisis y comparaciones.

5. Arquitectura del sistema

A continuación se va a describir la arquitectura del sistema. En primer lugar se enumerarán los componentes, y posteriormente se analiza cada uno por separado.

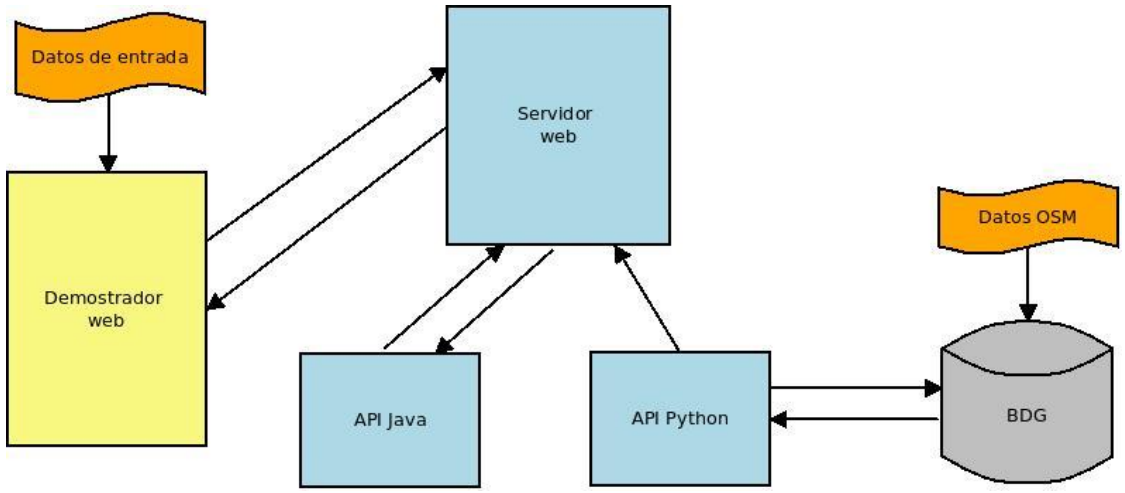

Figura 5.1 Arquitectura de el trabajo

Tal como se indica la Figura 5.1 los bloques que conforman este trabajo son lo siguientes:

- Datos geográficos de OSM.
- BDG.
- API Python.
- API Java.
- Demostrador web.

A continuación se va a explicar la funcionalidad y los componentes que conforman cada uno de los bloques del trabajo.

5.1. Datos geográficos de OSM

Este bloque está compuesto por los datos geográficos de entrada provenientes de OSM de una determinada región, en este caso de España. Estos datos servirán para conformar la BDG, que se explicará en el siguiente apartado.

5.2. Base de Datos Geográfica (BDG)

Este bloque ofrece los datos geográficos a la API Python. En este bloque se importan los datos geográficos de España provenientes de OSM mediante la herramienta de exportación *osm2pgsql* con el fin de conseguir el modelo de datos necesario para poder desarrollar la API.

5.3. API Python

Este es el bloque más importante del trabajo ya que es el encargado de transformar una coordenada geográfica o un recorrido en información. Esta información tiene que contener las coordenadas proyectadas en la calle más cercana, nombre de la calle, sentido y velocidad máxima entre otros datos.

Para la consecución de ese objetivo se puede dividir este bloque en dos bloques más pequeños con funciones más especificas.

- Conector con la base de datos: Se encarga de realizar las consultas necesarias a la BDG.
- Lógica de la API: Se encarga de aplicar la lógica para transformar las coordenadas de entrada en la información detallada anteriormente. Se comunica con el anterior bloque para obtener la información necesaria de la BDG.

5.4. Servidor web

El servidor web se realiza para cumplir con el siguiente objetivo. Se tiene que permitir el acceso a clientes a través de aplicaciones desarrolladas en cualquier lenguaje de programación distinto a Python, por ejemplo Java. Además de facilitar a otras aplicaciones trabajar con el servicio en remoto, sin necesidad de instalar los mapas de OSM en local junto con la aplicación final.

Por tanto la función de este bloque es la de implementar la funcionalidad citada en el bloque anterior mediante peticiones HTTP originadas tanto en otras aplicaciones como en el demostrador web, cuya funcionalidad se detallará posteriormente.

Este bloque recibe como datos de entrada peticiones HTTP. Que contienen un punto geográfico o una secuencia de los mismos, y devuelve la información proporcionada por la API Python.

5.5. API Java

Este bloque tiene la misma funcionalidad que el bloque de la API Python, con la única diferencia de que hace posible que se pueda implementar en aplicaciones Java.

5.6. Demostrador web

El demostrador web se implementa con la finalidad de comprobar y mostrar el funcionamiento de los bloques anteriores.

Para ello utiliza los bloques anteriores para representar y analizar los resultados obtenidos por la API. Se compone por dos principales bloques, tal como indica la Figura 5.2:

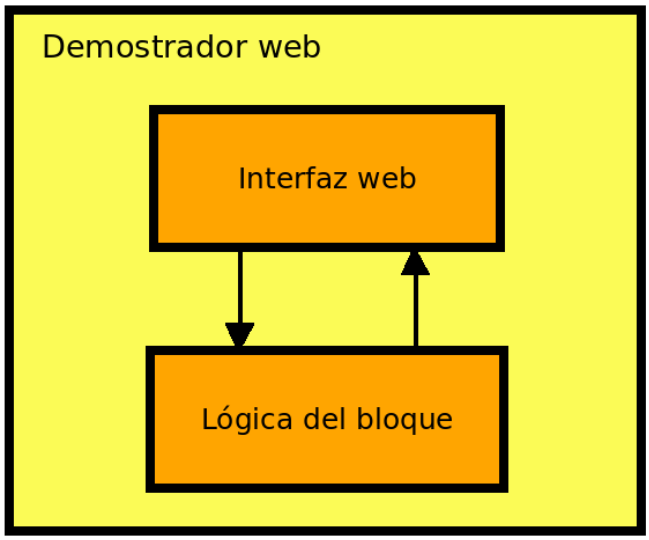

Figura 5.2 Arquitectura del demostrador web

- Interfaz web: Se representa un mapa interactivo en donde se pueden representar puntos y recorridos basándose en la API Python.
- Lógica web: Se encarga de procesar las peticiones del usuario y transmitirlas al servidor web mediante peticiones HTTP.

6. Diseño

Después de indicar la arquitectura de la aplicación se va a explicar al lector el diseño de los bloques comentados anteriormente, y las tecnologías elegidas para estos bloques.

La primera decisión que se tiene que afrontar se encuentra en los datos geográficos. Concretamente en el proveedor de datos geográficos para conformar la BDG. Se tiene que decidir entre Google Maps u OSM para este trabajo.

6.1. Google Maps u OSM

La principal y más importante diferencia entre Google Maps y OSM es el acceso que ofrecen a los datos de sus mapas. OSM es un proyecto con datos abiertos. Ofrecen sus datos geográficos de forma gratuita y con licencia libre. Estos datos se ofrecen con la estructura comentada en el apartado 2.

Esto supone que gran cantidad de desarrolladores, tanto en universidades como en grandes empresas, puedan ofrecer servicios y aplicaciones que no serían posibles sin contar con datos geográficos de OSM.

En contraposición Google Maps no ofrece sus datos geográficos, con el fin de mantener una ventaja comercial con los demás proveedores de mapas de datos geográficos. También protegen sus datos con diferentes derechos de autor y términos de restricciones de uso.

Por tanto aunque Google Maps se considere una plataforma de datos abierta, ya que ofrece una API [12] que cuenta con una versión gratuita para representar sus mapas en diferentes dispositivos y poder añadir marcadores y recorridos. Sin

embargo esta API gratuita tiene las siguientes limitaciones dependiendo del sistema operativo en el que se implemente:

- Android e IOS: Por defecto 1.000 solicitudes gratuitas por día, con un aumento de 150.000 solicitudes gratuitas por día después de validar la tarjeta de crédito.
- Web: Ofrece 25.000 visualizaciones del mapa diarias gratuitas. Se pueden conseguir 100.000 visualizaciones diarias extras a 0.50 $ cada 1.000 visualizaciones.

Además Google Maps no permite descargar sus datos geográficos al contrario que OSM, que con el archivo planet.osm ofrece los datos geográficos de todo el planeta. Como se ha comentado antes, está funcionalidad ofrecida por OSM permite el desarrollo de mejores servicios y aplicaciones geográficas.

Tanto Google Maps como OSM tienen sus mapas representados como vectores, que posteriormente representan visualmente para producir la vista del usuario. Sin embargo OSM posee la ventaja de representar visualmente los mapas al instante de realizar una corrección o modificación, lo que facilita contribuir en OSM. Esta funcionalidad no está permitida en Google Maps aunque por contraposición posee servidores más potentes y rápidos.

En conclusión OSM otorga el acceso a sus datos geográficos a los desarrolladores, lo que significa que se puedan desarrollar aplicaciones de mayor calidad con OSM que con Google Maps y con menos dificultades para conseguir los datos. Por tanto se escoge OSM como proveedor de datos geográficos para este trabajo.

Después de que se escoja el proveedor de datos geográficos para este trabajo se tiene que abordar a siguiente decisión de diseño, que consiste en decidir la forma de obtener los datos de OSM.

6.2. Origen de los datos geográficos de OSM

Como se ha explicado en el apartado de Estado del arte, encontramos dos formas para extraer los datos de OSM. La primera desde su página web, para

pequeñas regiones, o la segunda mediante terceros para regiones más grandes. Debido a que para este trabajo se necesita un BDG de España, se necesitará obtener los datos mediante terceros.

Tal como se indica en la figura 6.1, datos procedentes de la wiki de OSM. Hay varios proveedores de datos geográficos provenientes de OSM. En la siguiente imagen se indican estos proveedores con sus respectivas características.

Mirror	Area	Updated
http://download.geofabrik.de/	daily Shapefiles • entire continents (except North America) • all European countries • sub-country admin regions for France, Germany, UK • many countries outside Europe • States in the USA and Provinces in Canada	Daily
http://download.bbbike.org/osm/	More than 200 cities and regions worldwide or extract your own individual area (OSM, PBF, Garmin cycle map, Osmand, mapsforge, Navit and ESRI shapefile format)	Weekly
http://metro.teczno.com/	Individual metropolitan areas: • Shapefiles • Up-to-date coastline extracts	Monthly
Trimble Data Marketplace	Worldwide coverage: • Clip to any area you want - draw a polygon by hand or use a KML (1,000 vertice limit) to define your clip area • Choose any and all Map Features tags • Reproject • Supported formats: SHP, Esri FGDB, Safe FFS, GML, CSV, KML and DWG view quick walkthrough	Monthly
http://osmdata.thinkgeo.com/	Most countries in the world (shp file)	Weekly

Figura 6.1 Proveedores de datos OSM [7]

Por tanto después de que se analice la anterior comparativa, se elige como proveedor de datos geográficos provenientes de OSM a geofabrik.de, ya que nos ofrece los datos de España actualizados diariamente.

Con esta elección ya se disponen de los datos geográficos de OSM. Por tanto el siguiente paso es decidir con que sistema de gestión de base de datos conformaremos la BDG de España con los datos obtenidos de OSM a través de geofabrik.de.

6.3. Sistema de gestión de base de datos

Para el modelo de datos geográficos que ofrece OSM, se elige un modelo de base de datos relacional frente a uno no relacional. En cuanto al sistema de gestión de bases de datos se tienen en cuenta MySQL y PostgreSQL. Ambas son sistemas

de gestión de bases de datos libres, pero se opta por PostgreSQL por dos principales motivos:

- La existencia de herramientas de terceros tales como osm2pgsql, que permite importar datos geográficos de OSM a una base de datos PostgreSQL con una estructura diferenciada de puntos, calles y polígonos.
- La posibilidad de habilitar Postgis, una extensión que permite el uso de funciones geoespaciales que permite obtener y manipular mejor los datos geográficos.

Con la elección del sistema de gestión de gestión de datos queda totalmente terminado el diseño de la BDG de este trabajo. Por tanto a continuación se explica el diseño de la API Python, componente que conecta directamente con la BDG.

6.4. API Python

Para el diseño de la API se encuentran dos principales decisiones de diseño. La primera en la cual se elige el lenguaje para elaborar la API y la segunda, en donde se diseña el algoritmo mediante el cual se obtiene la calle más cercana.

La API se diseña con el objetivo de estar disponible tanto en Java como en Python. Sin embargo se decide desarrollar la API en un solo lenguaje de programación con el fin de disminuir el coste de desarrollo. Por tanto se elige entre Java y Python como lenguaje principal en la API.

1. Python o Java

Para realizar una correcta elección del lenguaje de programación elegido se realiza una comparación entre ambos:

- En cuanto al tipado del lenguaje: Python es un lenguaje de tipado dinámico, mientras que Java es un lenguaje con tipado estático.
- En cuanto al formato: En Python se emplea la indentación para separar el código en bloques. Esto permite desarrollar programas mucho más

claros y fáciles de leer. Sin embargo en Java se emplean llaves para separar el código.

- En cuanto al rendimiento: Java es más rápido que Python a la hora de ejecutar el código. Sin embargo este punto no es relevante para este proyecto ya que la parte más costosa de este trabajo es la búsqueda en la BDG, que no se hará ni en Java ni Python.
- En cuanto a simplicidad: Python es un lenguaje más simple lo que permite que sea más fácil desarrollar y aprender en él.

Teniendo en cuenta la comparación se elige Python para desarrollar la API principal, ya que permite un desarrollo más eficiente y rápido.

Después de justificar la elección de Python para desarrollar la API, se procede a diseñar el principal algoritmo de la API.

2. Algoritmo de la API

Antes de entrar en detalle sobre el diseño del algoritmo de la API se comentan brevemente las funcionalidades que este algoritmo debe conseguir. Se tiene que conseguir, partiendo de las siguientes entradas:

- Posición GPS actual del usuario.
- Posición GPS del usuario en el instante anterior, en el caso de que se disponga de esta posición.

A partir de estas entradas se tienen que conseguir la siguiente información de salida, conformada por los siguientes datos:

- Identificador de un segmento de vía.
- Posición proyectada del usuario sobre la vía en la que se encuentre.
- Sentido en el que circula el usuario dentro de la vía.
- Velocidad máxima de la vía por la que circula el usuario, en el caso de que esta información se encuentre en la BDG.

Para conseguir esta información de salida se barajan dos opciones:

- A partir de la posición GPS ofrecida por el usuario elegir la calle más cercana y a partir de ahí obtener todos sus datos.

- Algo más compleja que la primera. Se obtiene la calle más cercana de la posición anterior ofrecida por el usuario y se almacena en memoria. A continuación para la posición actual del usuario se obtienen las dos calles más cercanas, ya que se considera improbable que se encuentre en la tercera calle más cercana debido a la precisión de los terminales GPS actuales. A partir de estas dos calles se elige la que corresponda con la posición anterior, y en caso de no coincidir, implica que el usuario ha realizado un cambio de vía. Más adelante en el apartado de implementación se expone con más detalle la implementación de este algoritmo.

Después de valorar ambas alternativas, se opta por elegir la segunda, ya que aun siendo más compleja y más lenta, debido a que se necesitan un mayor número de consultas a la base de datos. Ya que es más precisa y da lugar a un menor número de errores.

Esta opción mejora los resultados en intersecciones de dos vías, donde debido a la precisión del GPS se pueda elegir una vía donde no se encuentre el usuario

Después de explicar tanto los datos usados como el algoritmo utilizado, se procede a diseñar un servidor web que sirva para elaborar un demostrador, en el cual se puedan realizar las pruebas pertinentes para mostrar el correcto funcionamiento de la aplicación. Además este servidor también tiene que permitir el uso de la API mediante llamadas HTTP.

6.5. Servidor web

Para el diseño del servidor web y poder importar el módulo de la API Python se consideran dos *frameworks*, como Flask y Django, los cuales se comparan a continuación para justificar la elección de uno de ellos.

Antes de entrar en detalle sobre que *framework* web utilizar, es conveniente explicar en qué consiste un *framework* web. Un *framework* web es una biblioteca de paquetes o módulos que facilitan crear aplicaciones web escalables, fiables y fáciles de mantener. También permiten reutilizar el código para operaciones HTTP con mayor facilidad.

Después de conocer qué es y cual es el uso de un *framework* web se comparan dos de los principales *frameworks* web en Python.

1. Flask o Django

Para elegir entre uno de estos dos *frameworks* se analizan las fortalezas y las debilidades de ambos, primero se empieza con Django. Django facilita a los desarrolladores de Python elaborar aplicaciones web de forma rápida sin necesidad de pensar previamente en la infraestructura de la aplicación. Permite desarrollar mejores aplicaciones web, de forma rápida y con un menor número de líneas de código. Además cuenta con una excelente documentación, y una gran comunidad de desarrolladores. En cuanto a las ventajas de usar Django se encuentran:

- Software bien elaborado, que cuenta con muchos *plugins*.
- Gestión de base de datos simple.
- Soporta mapeo objeto-relacional.
- Altamente personalizable.

Sin embargo también se encuentran desventajas:

- Se diseña para grandes proyectos, para realizar pequeños proyectos puede no ser eficiente.
- Cuenta con una curva de aprendizaje bastante lenta.

Sin embargo Flask se puede considerar el contrario de Django. Flask es un *framework* web relativamente joven, se usa desde 2010. Flask es el *framework* web perfecto para iniciarse en construir una aplicación web debido a las facilidades que proporciona. Se pueden citar sus ventajas y desventajas, tal como se hizo con Django anteriormente. En cuanto a sus ventajas encontramos:

- Fácil de aprender y utilizar
- Minimalista sin sacrificar potencia.
- *Framework* web flexible y fácilmente extensible.

Sin embargo también se encuentran desventajas:

- Cuenta con una documentación bastante limitada.
- Falta gestión de base de datos y no soporta mapeo objeto-relacional.

- Está bastante limitado.

Después de analizar ambos *frameworks* web se opta por elegir Flask, debido a su fácil aprendizaje y a la baja complejidad de esta aplicación. Con esto se reduce el coste de desarrollo de este servidor web.

Al conocer el *framework* web que se utiliza para realizar éste trabajo. Se explica el diseño y las funcionalidades que ofrece el servidor web.

2. Servidor web

Se decide ofrecer al usuario todo tipo de opciones para obtener información de la API a través de el servidor web. Esto implica que es posible tanto recibir información de puntos GPS a través de peticiones HTTP simples a una URL como el uso un demostrador web, que proporciona una interfaz con la cual se interactúa con el mapa para recibir coordenadas o subir ficheros con recorridos de un vehículo.

Con estas funcionalidades se asegura que el usuario pueda realizar cualquier tipo de operación a través de este servidor web. Después de conocer el diseño del servidor web, se va a cerrar este apartado comentando el demostrador web de esta aplicación.

6.6. Demostrador web

El demostrador web tiene que cumplir la función de realizar pruebas unitarias y de recorridos. También ofrece la posibilidad de interactuar con el mapa. Por tanto el diseño de este módulo se centrará en la interfaz y los componentes de ésta.

Se elige un diseño muy simple, tal como se indica en la figura 6.2, que consta de el mapa interactivo y 3 botones para realizar pruebas, estos botones son:

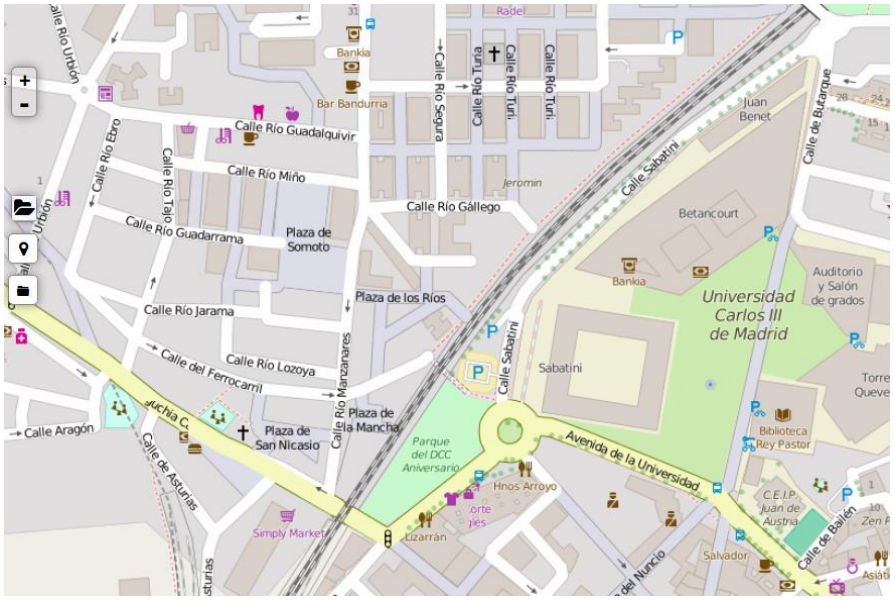

Figura 6.2 Interfaz del demostrador web

- Botón con dos posiciones que sirve para habilitar y deshabilitar las funciones interactivas con el mapa para simular un recorrido pulsando en el mapa. Las posiciones son las siguientes:

    - Posición 1: Se usa para habilitar la función de interactuar con el mapa punto a punto pasando por la API diseñada anteriormente. Se representan en el mapa los puntos resultantes, tal como se indica en la figura 6.3, y en caso de haber ya puntos representados se borran todas las marcas al pulsar el botón.

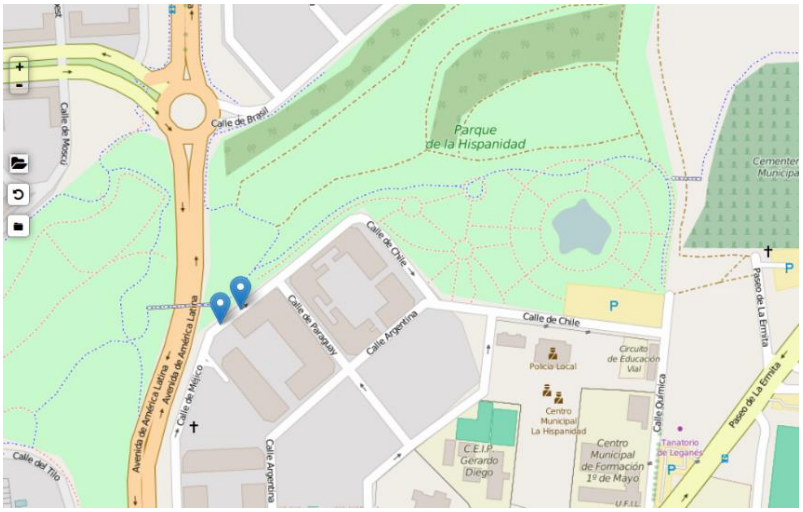

Figura 6.3 Botón para interactuar pulsado

- Posición 2: Deshabilita la función de interactuar con el mapa. La interfaz se diseña tal como se indica en la figura 6.4.

Figura 6.4 Botón para interactuar sin pulsar

- Botón que permite cargar un recorrido en formato .kml o .csv. Después de pasar por la API, se representará el recorrido en el mapa.
- Botón para representar directamente el recorrido de un archivo .kml. Este botón se añade para realizar pruebas y validaciones posteriormente.

Con esto queda cerrado el diseño de este trabajo. Posteriormente se va a explicar los aspectos más relevantes en la implementación de este trabajo.

7. Implementación

En este apartado se explicará al lector los aspectos más importantes en la implementación de este trabajo.

En primer lugar se explicarán las consultas a la BDG, como se construyeron y que información obtienen.

7.1. Consultas a la BDG

Para consultar los datos en la BDG encontramos dos tipos de consultas, las cuales se explican a continuación. Primero se explicarán los objetivos que deben cumplir y posteriormente los campos que se necesitan obtener y las restricciones para obtener los resultados.

1. Consulta 1: *Reverse geocoding*

En esta consulta se tiene que cumplir el objetivo de realizar un correcto *reverse geocoding*. Antes de entrar en detalle es conveniente explicar en qué consiste el *reverse geocoding*. Es el proceso por el cual a partir de un par de coordenadas geográficas (latitud y longitud) se obtiene una dirección o cualquier lugar en la que se encuentren dichas coordenadas.

Por tanto para cumplir el objetivo de este trabajo se necesitan obtener los datos en la consulta de la tabla planet_osm_line :

- Osm_id : Id que representa la calle, punto o polígono en los datos geográficos de OSM. En esta consulta es el campo más importante a obtener ya que es el identificador de la calle y a través de este se pueden obtener el resto de atributos de la calle

- Coordenadas proyectadas: Coordenadas geográficas proyectadas sobre la calle. Esta información es necesaria para representar puntos y recorridos en el demostrador web, para ello se aplicarán los siguientes métodos de Postgis, tal como se indica en su documentación [13]:
    - ST_AsText : Método que devuelve la representación *Welll-Known Text* (WKT) de la geometría o geografía dada sin los metadatos del *Spatial Reference System Identifier* (SRID).
    - ST_Transform: Método que a partir de una geometría y un SRID devuelve una nueva geometría transformada al sistema de referencia espacial que corresponde con el SRID. Este SRID tiene que existir en la tabla SPATIAL_REF_SYS. (IMAGEN)
    - ST_Line_Interpolate_Point: Método que a partir de la geometría de la línea y el porcentaje de la línea recorrida, entre 0 y 1, devuelve el punto interpolado en la línea.
    - ST_LineLocatePoint: Método que a partir de un punto y una línea devuelve un decimal entre 0 y 1 que representa la localización del punto más cercano en la calle del punto que se pasa por parámetro. Este método usado junto al método ST_Line_Interpolate_Point devuelve el punto de la calle más cercano a un punto dado, también se usa para calcular el porcentaje recorrido de una calle.
    - ST_SetSRID: Método que establece o edita el SRID de una geometría a un número entero. La correspondencia del número entero con el SRID se encuentre en la tabla SPATIAL_REF_SYS.
    - ST_Point: Constructor que crea la geometría de un punto a partir de coordenadas geográficas (latitud, longitud).

Al juntar los métodos anteriores se puede conocer el punto proyectado en la calle con el siguiente fragmento de la consulta final, donde *way* es la geometría de la calle y lat y lon las coordenadas geográficas:

"Select osm_id,
st_asText(st_transform(ST_Line_Interpolate_Point(way,ST_Line_Loca
te_Point(way,ST_SetSRID(ST_Point("+lon+","+lat+"),4326))),4326))
from planet_osm_line"

Después de explicar los datos que se necesitan obtener en la consulta, se explicará cómo acceder a las dos calles más cercanas a partir de las coordenadas geográficas de un punto. Se aplican el siguientes método de Postgis:

- ST_DWithin: Método que devuelve *true* si dos geometrías están a la distancia indicada como parámetro. Esta distancia se mide en unidades del sistema de referencia espacial de las geometrías.

Con el método anterior junto con otros explicados anteriormente tales como ST_SetSRID y ST_Point se puede conformar la restricción de la consulta. Se tiene que tener en cuenta también que para encontrar calles el campo *highway* tiene que existir. Por tanto la clausula *where* queda:

"where ST_DWithin(ST_SetSRID(ST_Point("+lon+","+lat+"),4326), way, "+str(dist)+") and highway is not null"

Para terminar de explicar la consulta, se explican los parámetros necesarios para realizarla:

- Coordenadas geográficas: Se definen por un par longitud, latitud que se obtienen por el GPS del usuario, que corresponden con su posición.
- Distancia: Parámetro que se usa para determinar el radio de búsqueda del método Postgis ST_DWIthin. Para conseguir que esta consulta siempre devuelva resultados, se implementa un aumento incremental de este parámetro por si la consulta no devuelve resultados.

Con el fin de ilustrar la consulta se simula en el demostrador web las dos vías más cercanas a un punto geográfico dado, tal como indica la figura 7.1.

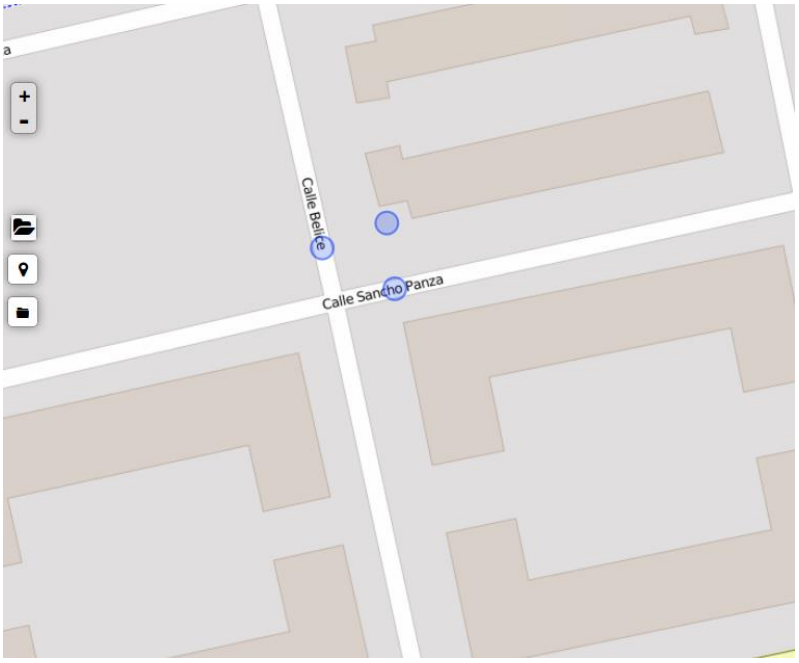

Figura 7.1 Calles más cercanas a unas coordenadas geográficas

En donde la consulta que se emplea para el ejemplo anterior devuelve los siguientes resultados, tal como se indica en la figura 7.2.

	osm_id bigint	name text	st_astext text	st_distance double precision	st_astext text
1	121842074	Calle Sancho Panza	LINESTRIN	119418980218285	POINT(-3.780
2	31453835	Calle Belice	LINESTRIN	159377619220541	POINT(-3.780

Figura 7.2 Resultados de la consulta ejemplo

Después de explicar la consulta a la base de datos con la que se consigue el *reverse geocoding* de la aplicación. A continuación se explica la implementación de la consulta con la que se obtienen los parámetros de la calle en la que se encuentra el usuario.

2. Consulta 2: Detalles de la vía

Esta consulta es bastante más simple que la anterior, ya que únicamente se obtienen los parámetros a partir de un identificador de OSM. Los parámetros que se obtienen son los siguientes:

- Nombre: Nombre de la calle en la que se encuentra el usuario.
- Superficie: Superficie de la calle en la que se encuentra el usuario.
- Velocidad máxima: Velocidad máxima de la calle en la que se encuentra el usuario. Se encuentra en el campo *tags* de la base de datos, en el que se encuentran datos relevantes de la calle. Se obtienen de la siguiente forma para, por ejemplo para la velocidad máxima:

  "tags->'maxspeed'"

La tabla es la misma que en la consulta anterior, "planet_osm_line" ya que en ella se encuentran todas las calles y vías que conforman la base de datos. Por tanto la consulta final quedaría:

"select name,surface,tags->'maxspeed' from planet_osm_line where osm_id="+osm_id"

Después de comentar la implementación de las consultas a la base de datos se continúa con otros aspectos relevantes para la implementación de este trabajo, como es el diseño del algoritmo con el cual a partir de estas consultas obtener la calle más cercana.

7.2. Algoritmo de geo posicionamiento

Para conseguir obtener la calle en la que se encuentra el usuario, en vez de elegir la calle más cercana se implementa un algoritmo para ello.

Como se explicó en el apartado de diseño el algoritmo se basa en las dos posiciones más cercanas para disminuir el error al decidir la calle mas cercana. Para implementar el algoritmo se utilizan las consultas explicadas en el anterior apartado con el fin de obtener las dos calles más cercanas a la posición ofrecida por el usuario. También se necesita la posición anterior, que será la que ofreció el usuario

anteriormente, a no ser que sea el primer punto que se envía a la aplicación implementada.

Con toda la información relativa a la posición del usuario ya se puede implementar el algoritmo de geo posicionamiento de este trabajo. Para ello se pueden encontrar los siguientes casos:

1. Caso 1: Solo se encuentra una calle cercana

En el caso de que la consulta con la que se obtienen las calles más próximas solo devuelva un resultado, en vez de dos, se considera que no hay otra calle suficientemente próxima para elegirla, tal como se indica en la figura 7.3.

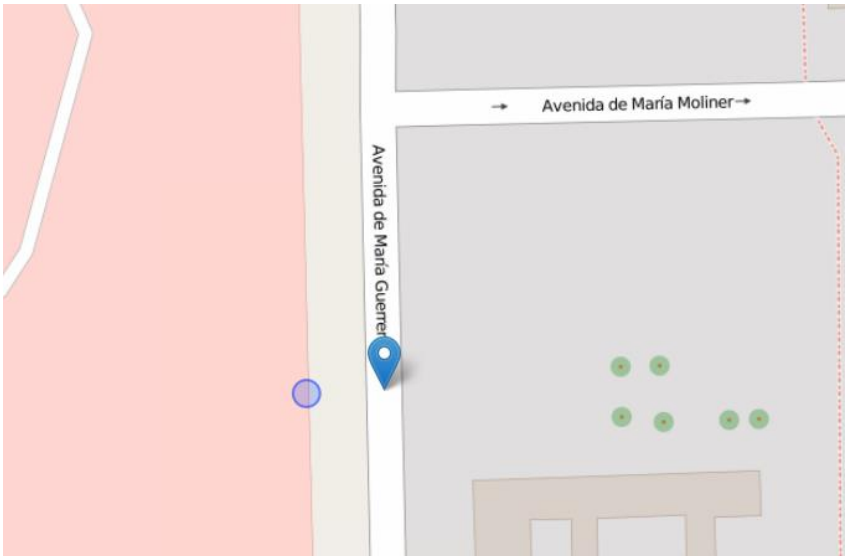

Figura 7.3 Caso donde solo se encuentra un resultado

Por tanto como se observa en la figura 7.3 al no haber otra calle cercana se elige ésta como resultado.

2. Caso 2: Se encuentran dos calles cercanas

Este es el caso más común, ya que si el usuario se encuentra en una ciudad con gran facilidad la consulta del apartado anterior devolverá dos resultados. Dentro de este caso se encuentran dos alternativas, dependiendo de si se conoce o no la posición anterior del usuario:

- En caso de que no se disponga la posición anterior del usuario, se decidirá por la calle más próxima al usuario.
- En caso de que si se disponga de la posición anterior del usuario, se decidirá el resultado que concuerde con la posición anterior, así se disminuye el error de decisión.

A continuación, en la figura 7.4 se muestra un ejemplo de una sucesión de dos puntos teniendo en cuenta nuestro algoritmo y posteriormente en la figura 7.5 donde se tiene en cuenta únicamente la posición más cercana.

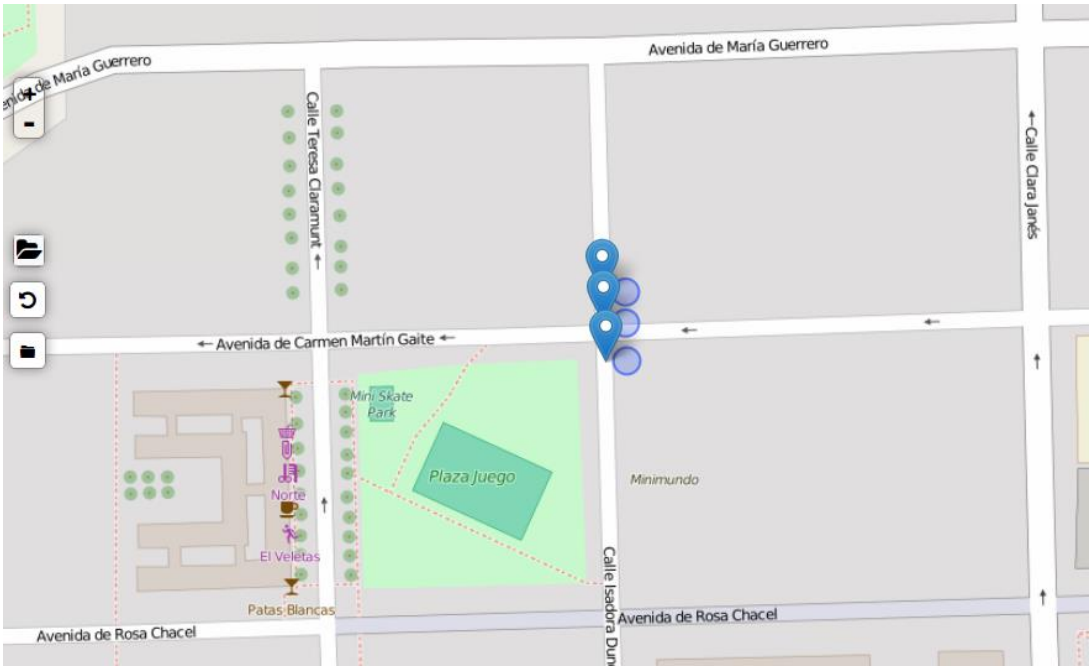

Figura 7.4 Sucesión de tres puntos con el algoritmo de este trabajo

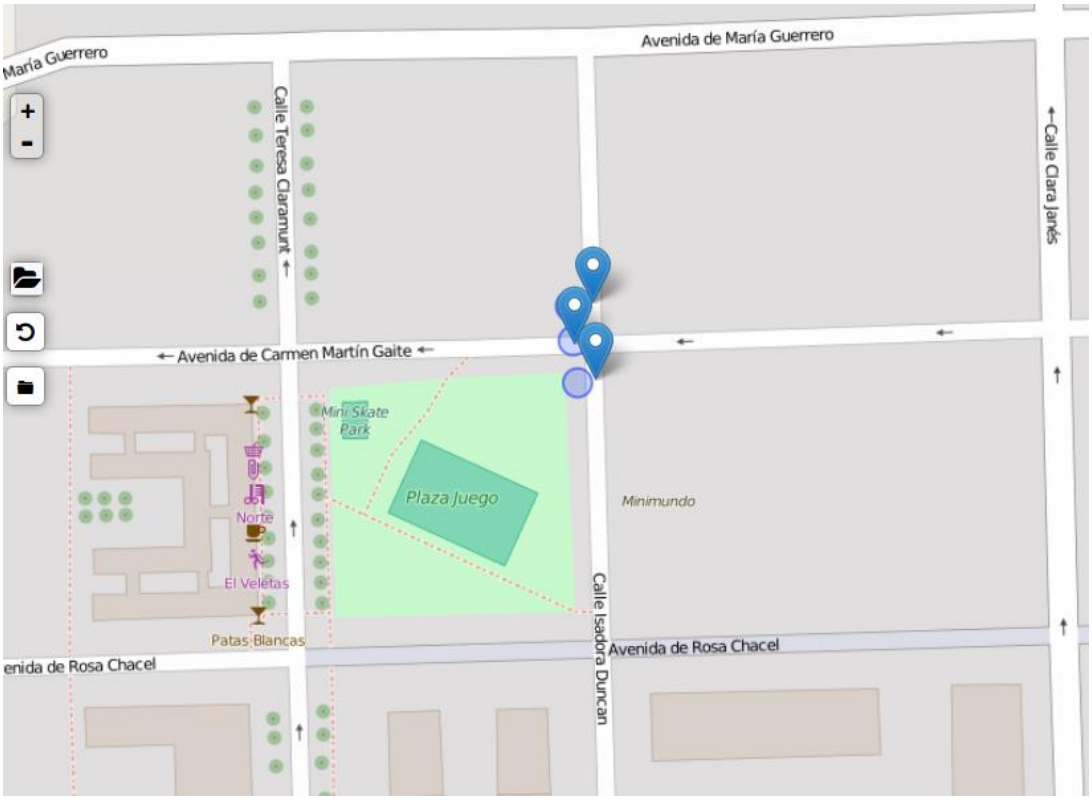

Figura 7.5 Sucesión de tres puntos con el algoritmo de la calle más cercana.

Tras comparar la figura 7.4 con la figura 7.5 se observa que en la figura 7.4 empleando el algoritmo usado en esta aplicación se producen menos errores que en el caso de la figura 7.5, al simular el trayecto de un vehículo al atravesar un cruce. Por tanto se puede considerar que el algoritmo usado en esta aplicación es más robusto y comete menos errores que el algoritmo de la calle más cercana.

Por tanto tras cubrir todos los casos posibles del algoritmo de geo posicionamiento se puede resumir el funcionamiento tal como indica el diagrama de flujo de la figura 7.6.

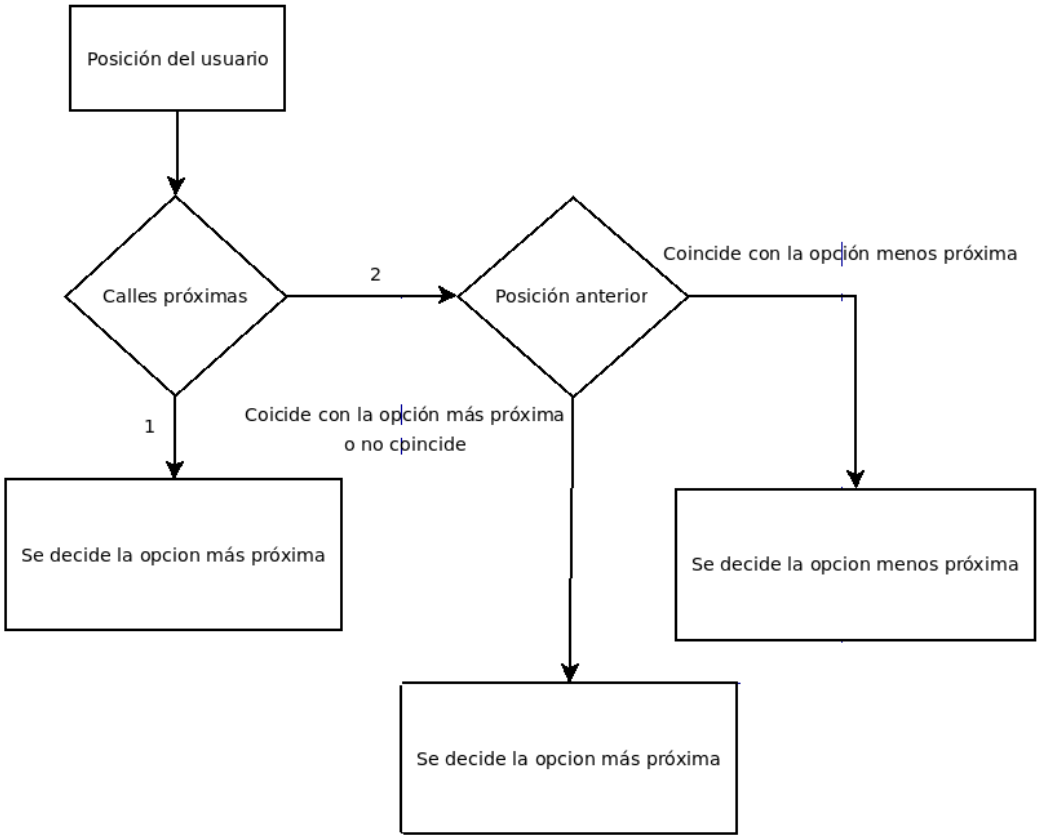

Figura 7.6 Diagrama de flujo del algoritmo de geo posicionamiento

A continuación después de acabar con la implementación del algoritmo de geo posicionamiento se procede a comentar aspectos importantes en la implementación del servidor web.

7.3. Servidor web

En cuanto a la implementación del servidor web, primero es conveniente explicar la función de este bloque. La principal función del servidor web es integrar la API para que funcione mediante llamadas HTTP.

Esta funcionalidad supone la posibilidad de llamar la API desde cualquier navegador mediante llamadas HTTP e integrar el demostrador web para realizar pruebas unitarias y de recorrido de la API.

Por tanto se implementan dos principales rutas para el servidor web, la primera para integrar el demostrador web para un solo punto geográfico y la segunda para implementar recorridos de varios puntos geográficos.

1. Ruta 1: Para un único punto geográfico

El procedimiento que se sigue en esta ruta para recopilar el punto geográfico a través de una llamada HTTP y posteriormente devolver un resultado consta de tres principales etapas:

1. Se obtienen las coordenadas geográficas del punto, estas coordenadas se pasan como parámetros en la *Uniform Resource Locator* (URL) con sus correspondientes etiquetas. Para ello se pueden tener dos casos dependiendo de si se pasan las coordenadas del punto anterior o no:
   i. Solo se pasa como parámetros las coordenadas actuales del usuario el cual llama a la API de este trabajo. Se utilizan las etiquetas *lat* y *lon* para definir la latitud y longitud en las que se encuentra el usuario. Por tanto la URL sigue el siguiente formato:

      “url = "punto_mapa?lat="latitud del usuario "&lon="+longitud del usuario;”

   ii. Se pasan como parámetros tanto la posición actual como la posición anterior del usuario. Para ello se utilizan las etiquetas *lat* y *lon* para las coordenadas en las que se encuentra actualmente el usuario, y *lat_ant* y *lon_ant* para las coordenadas en las que se encontraba anteriormente el usuario. Por tanto la URL sigue el siguiente formato:

      “url = "punto_mapa?lat="latitud del usuario "&lon="longitud del usuario "&lat_ant="latitud anterior "&lon_ant="longitud anterior;”

2. Con las coordenadas geográficas ya obtenidas solo necesitamos importar las librerías de nuestra API en el servidor web y posteriormente utilizar el algoritmo con el cual se obtiene una respuesta a través de las coordenadas actuales y anteriores del usuario que llama esta aplicación.

3. Después de obtener la respuesta, ésta se transforma al formato *Geographic JavaScript Object Notation* (GeoJSON) ya que con este formato se puede utilizar en el demostrador web para representar los puntos geográficos finales devueltos por la API.

Después de aplicar estos tres pasos se puede determinar la posición del usuario después de ser procesada por la API de este trabajo en un demostrador web o en cualquier navegador mediante una respuesta HTTP. Por tanto solo queda por explicar la implementación de la ruta con la cual se pueden procesar recorridos en el servidor web.

2. Ruta 2: Para un recorrido de puntos geográficos

En esta ruta se trata un conjunto de puntos geográficos al contrario que en la anterior ruta. Para ello se recibirán un conjunto de puntos y posteriormente se obtiene como resultado el mismo conjunto de puntos después de pasar por la API de este trabajo.

Para ello el procedimiento que se sigue es similar al de la anterior ruta y consta de tres principales etapas:

1. Se obtiene el conjunto de coordenadas geográficas. A diferencia de la anterior ruta en ésta el conjunto se pasa mediante un POST al servidor web con un fichero. Este fichero se puede pasar en dos formatos:
   i. Formato *Comma-separated values* (CSV): En el cual cada línea corresponde a un punto geográfico del usuario, en el que latitud y longitud están separados por comas.
   ii. Formato *Keyhole Markup Language* (KML): En este formato cada punto geográfico del usuario va acompañado de más información. Por ejemplo cada punto geográfico contiene las siguientes etiquetas en el formato KML:

“ <Placemark>
<name></name>
<description></description>
<Point>
<coordinates>-3.7592027,40.3438303,0</coordinates>

</Point>
</Placemark>”

Por tanto de esa línea se extrae las coordenadas del punto para su posterior procesamiento.

2. Después de que se obtengan el conjunto de coordenadas geográficas procedentes del fichero introducido por el usuario se tienen que procesar los puntos geográficos introducidos por el usuario con su posición anterior por el algoritmo de la API. Para ello se almacena en memoria la posición anterior del usuario y se van añadiendo los resultados a una lista para posteriormente devolver la información al demostrador web.
3. Para terminar el procesamiento del fichero simplemente queda por explicar el formato que tiene cada punto resultante. Se utiliza el formato geoJSON para que el demostrador web los pueda interpretar correctamente. Este formato consta de tres partes que son las siguientes:
   i. Tipo: Con la etiqueta *type*, en este trabajo irá por defecto a “Feature”. Significa que se trata solo de una geometría, en el caso de ser varias se utilizaría la etiqueta "FeatureCollection".
   ii. Propiedades: La etiqueta *properties* contiene información relativa al punto. En este trabajo contiene la información relativa al identificador OSM y el sentido en esa calle.
   iii. Geometría: La etiqueta *geometry* contiene el tipo de geometría del objeto (línea, punto o polígono) con la etiqueta *type* y las coordenadas de la geometría con la etiqueta *coordinates*.

Por tanto el formato final de un punto geográfico queda con el siguiente formato:

“"type": "Feature",

"properties": {

"osm_id": ‘OSM id del punto resultante’

"sentido": ‘Sentido del usuario en la calle (I->F o F->I’ },

"geometry": {

"type": "Point",

"coordinates": 'Coordenadas del punto resultante' }"

Para terminar se envía la lista con el conjunto de resultados como JSON al demostrador web para que posteriormente éste los pueda interpretar. En el siguiente apartado se comenta los aspectos más importantes en la implementación del demostrador web.

7.4. Demostrador web

El demostrador web se utiliza para realizar pruebas de la API representando los resultados en un mapa para facilitar el análisis de los resultados. Para implementar este demostrador se utiliza Leaflets, una librería JavaScript para crear mapas interactivos, en la cual se utiliza su documentación [14]. La implementación de este bloque la podemos dividir en cuatro partes, que son el mapa, los marcadores, los eventos y los botones.

1. El mapa

Es el componente más importante en el demostrador web, ya que sobre él se interactúa y en él se encuentran los botones y los marcadores. Para crear el mapa en el demostrador web se necesita utilizar los siguientes métodos, los cuales se indican en la documentación de Leaflets [14]:

- L.map: Instancia el mapa en un *div HyperText Markup Language* (HTML), que puede ser definido por su id. También se le pueden pasar opciones para su creación, que se modifican con el siguiente método:
    - setView: Establece las coordenadas geográficas en las que se muestra el mapa en el demostrador y el zoom con el que se muestra.
- L.tileLayer: Establece la fuente de los datos en la cual se representa el mapa y algunas opciones como puede ser el *zoom* máximo que se puede establecer en el mapa.
- AddTo: Se usa con el fin de añadir opciones al mapa, en este caso se añade la fuente de los datos al mapa.

Después de juntar ambos métodos explicados anteriormente se utiliza para construir el mapa las siguientes líneas:

```
var map = L.map('map').setView([40.34386902, -3.75917351], 14);

var tile = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
maxZoom: 19, attribution: 'Map data &copy; <a
href="https://openstreetmap.org">OpenStreetMap</a>' }).addTo(map);
```

El resultado de aplicar esos comandos en el demostrador web es el indicado en la figura 7.7 .

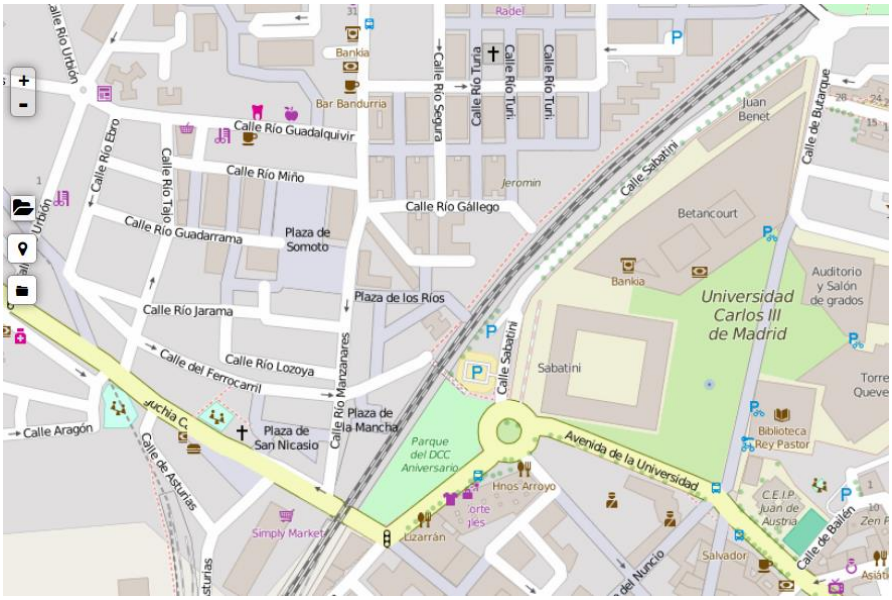

Figura 7.7 Mapa de OSM en el demostrador web

Después de explicar cómo se implementa el mapa en el demostrador web se continúa con la implementación de los marcadores en el mapa.

2. Marcadores

Después de conocer cómo se instancia el mapa en el demostrador web, es necesario conocer cómo se interactúa y se representan los datos en el mapa. En el demostrador web de éste trabajo se utilizan dos tipos de marcadores, para

diferencias pruebas unitarias de pruebas de recorrido. Estos marcadores son los siguientes:

- L.marker: Instancia un marcador en el mapa en las coordenadas geográficas que se pasan como parámetro. También se le puede añadir opciones, como por ejemplo un mensaje emergente con el siguiente método:
    - bindPopup: Añade contenido HTML en un mensaje emergente al marcador desde el que se llama.

  En este trabajo un marcador de este tipo con mensaje *popup* se representa como se indica en la figura 7.8.

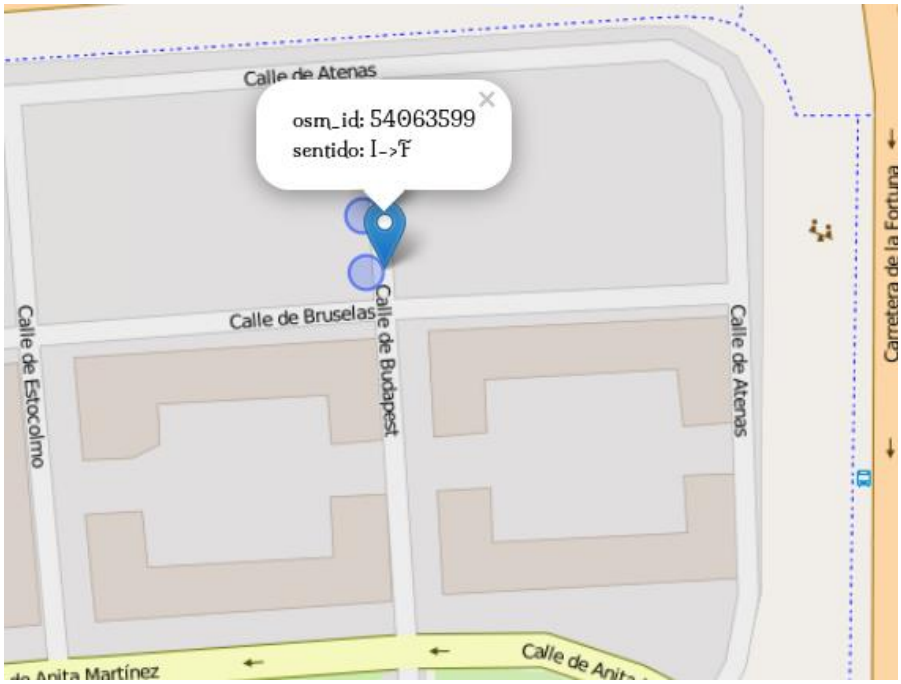

Figura 7.8 Representación de un marcador L.marker con mensaje *popup*

- L.circleMarker: Instancia un marcador circular en las coordenadas geográficas que se pasan como parámetro. Al igual que L.marker se le pueden añadir opciones y añadir mensaje *popup*. Gráficamente este marcador se representa tal como se indica en la figura 7.9.

Se utilizan dos tipos de marcadores para diferenciar gráficamente rutas punto a punto de rutas pasadas por fichero. Usándose el marcador L.marker para rutas punto a punto y el marcador L.circleMarker para ficheros.

3. Eventos

Tras comentar qué tipo de marcadores se implementan en este trabajo se procede a explicar cómo se manejan los eventos en el demostrador web. Estos eventos principalmente son obtener las coordenadas al interactuar con el mapa y transmitir y representar ficheros en el demostrador web.

Para realizar la obtención coordenadas se usan métodos proporcionados por Leaflet y JavaScript [15]. En cuanto a los métodos que ofrece Leaflet se usa el siguiente:

- On: Añade un escuchador al objeto desde el que se llame. Como parámetro se le pasa el tipo de acción a escuchar y la función JavaScript que se ejecutará cuando se produzca la acción. Para este demostrador web como parámetro se pasa '*click*' y la función 'obtener_coords' cuya implementación se explicará posteriormente.

Después de que se explique cómo se añade el *listener* al mapa del demostrador web se continúa con la implementación de la función 'obtener_coords' cuya función será obtener las coordenadas en la que se ha pulsado en el mapa y mandar esa información al servidor web y manejar posteriormente el resultado que el servidor devuelva. Para conseguir ese objetivo se utilizan las siguientes funciones:

- e.latlng.lat y e.latlng.lon: Con estas funciones se obtiene la latitud y longitud en las que el usuario ha interactuado con el mapa. Con ellas se conforma la URL por la cual se accede al servidor web, en la ruta 'punto_mapa', que se describió en la implementación del servidor web. La URL conformada es la siguiente:

  "url = "punto_mapa?lat="+e.latlng.lat+"&lon="+e.latlng.lng;"

- $.get: Método JavaScript que realiza una petición HTTP GET a la URL que se pasa como parámetro, en este caso se hace al servidor web para obtener el punto resultante tras procesar por la API el punto pulsado por el usuario.

Como parámetro se le pasa la URL a la que realizar la petición, la función en la que posteriormente se procesa la respuesta y el tipo de datos que se espera del servidor. Por tanto el comando es el siguiente:

"$.get(url, marca_api, "json")"

Finalmente en la función 'marca_api' se añade un marcador a las coordenadas geográficas resultantes añadiendo un mensaje emergente con información sobre el osm_id y el sentido de la vía.

En cuanto al evento de leer un fichero se diseña una ventana en la cual cargar el fichero mediante un formulario. Para su envío al servidor web se emplea el siguiente comando JavaScript:

- $.ajax: Método JavaScript que sirve para realizar peticiones HTTP asíncronas. Para ello como parámetro se pasa los siguientes:
  - Tipo: Con la etiqueta *type* se indica el tipo de petición a hacer a la URL. En este caso al tener que enviar datos al servidor se trata de una petición HTTP POST.
  - URL: Dirección a la que se realiza la petición HTTP POST. En este caso se hace a la ruta 'upload_ajax' que procesará el fichero enviado y posteriormente enviará una respuesta.
  - Datos: Con la etiqueta *data*, se le pasa los datos que se obtienen en el formulario.
  - Tipo de datos: Con la etiqueta *dataType*, se indica el tipo de datos a esperar en la respuesta del servidor. En este caso es 'json'.

Se utiliza el comando $.ajax para subir el fichero asíncronamente y por tanto no bloquear el demostrador web durante la subida del fichero al servidor para su procesamiento.

Para procesar la lista de puntos resultantes que ofrece el servidor web, se emplea el siguiente comando que ofrece Leaflet:

- L.geoJson: Crea una capa GeoJSON. Por parámetro se pasa un *array* en formato GeoJSON que es el ofrecido como respuesta en el servidor y el estilo de los marcadores para representar los resultados.

Con esto ya queda explicada la implantación de los eventos del demostrador web. A continuación se trata sobre la implementación de los botones necesarios en el demostrador web.

4. Botones

La función de los botones en el demostrador web es la de habilitar funciones y acciones en el mapa. Se implementan dos botones:

- El primero con dos posiciones, que tiene dos funciones la de habilitar y deshabilitar la posibilidad de interactuar en el mapa y la de gestionar las capas que se representan en el mapa. Para su implementación se implementa un variable que cambie según si el botón se encuentra en la posición para interactuar en el mapa o no. En cuanto a la gestión de capas en el mapa se emplea el siguiente componente de Leaflet:
    - L.layerGroup: Crea una capa que sirve para agrupar marcadores y permite usar funciones para añadir las capas al mapa y posteriormente borrarlas.

  Dependiendo de la posición en la que se encuentre el botón se emplean distintos métodos en las capas del mapa. Si el botón se encuentra en la posición para interactuar con él, se emplea el siguiente método:

    - Si se encuentra en la posición que permite interactuar con el mapa se emplea el método addLayer para añadir la capa al mapa y añadir cada punto pulsado a la capa que conforman todos los marcadores pulsados.
    - Si se encuentra en la posición que no lo permite se emplea el método removeLayer para borrar todos los marcadores anteriores y posteriormente el método L.layerGroup() para crear una nueva capa vacía para posteriormente añadir marcadores.
- El otro botón tiene la función de abrir un formulario para subir un fichero al servidor y posteriormente representarlo en el mapa. También se usa el método addLayer para añadir la capa de los puntos resultantes del fichero al mapa.

Ambos botones se crean mediante un constructor proporcionado por las librerías de EasyButton [16] que es el siguiente:

- L.easyButton: Crea un botón dentro del mapa pasándole ciertos parámetros, que son:
    - States: Indica las posibles posiciones que puede tener el botón. Se indica como se fueran un array.
    - Icon: Icono del botón se usan como proveedores de iconos EasyButton y Ionicons [17]
    - Statename: En el caso de que el botón tenga varios estados indica el nombre del estado del botón.
    - Tittle: Indica el titulo del botón, que saldrá al ponerse encima de el botón.
    - OnClick: Se usa para definir la función que se ejecuta al pulsar el botón.

Al terminar de comentar la implementación de los botones queda cerrada la sección de implementación de este trabajo. A continuación se van a explicar las pruebas que se realizan en este trabajo y se analizan los resultados.

8. Validación/pruebas

Para probar que la API funciona correctamente y presenta un bajo número de errores se realizan pruebas unitarias y de recorrido. En ellas se pretende comprobar que la API funciona correctamente por lo que se probarán varios casos posibles.

Como se ha dicho anteriormente se realizarán dos tipo de pruebas, unitarias y de recorrido, dependiendo del modo en el que se realicen éstas en el demostrador web.

8.1. Pruebas unitarias

Primero se realizan pruebas unitarias de la aplicación en las cuales se prueban todos los posibles casos que se pueden dar en la API y se comprobará si esta funciona correctamente.

Para ello se valida cada caso en el demostrador, luego se explica el funcionamiento de cada caso y posteriormente se analizan los resultados. En las pruebas en el demostrador web se indica con un círculo azul la posición del usuario y con un marcador la posición resultante.

1. Caso 1: Solo se encuentra una calle cercana

Es el caso más simple que se puede comprobar, ya que consiste en probar si en un punto que solo posea una calle cercana, se elige esta calle como resultado final, se simula esta situación en el demostrador web y se obtiene el resultado mostrado en la figura 8.1.

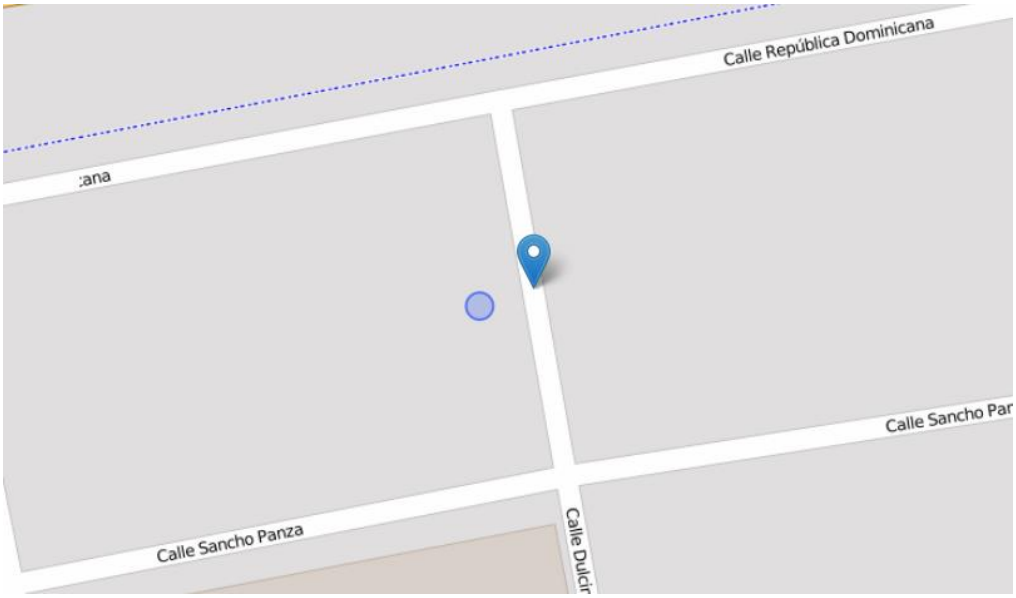

Figura 8.1 Prueba unitaria para el caso 1

Como se observa en la figura 8.1 pese a ser un caso muy fácil se comprueba que funciona correctamente y por tanto se puede continuar con otro caso.

2. Caso 2: Se encuentran dos calles cercanas, pero sin posición anterior

Más común que el anterior, ya que sobre todo en ciudad se encuentran dos calles cercanas con alta frecuencia en situaciones como cruces o dos calles paralelas muy cercanas. En estas situaciones al no poseer información de la posición anterior se tiene que elegir como resultado la calle más cercana. Por tanto se simula en el demostrador web esta situación mediante un punto próximo a un cruce y se obtiene el resultado mostrado en la figura 8.2.

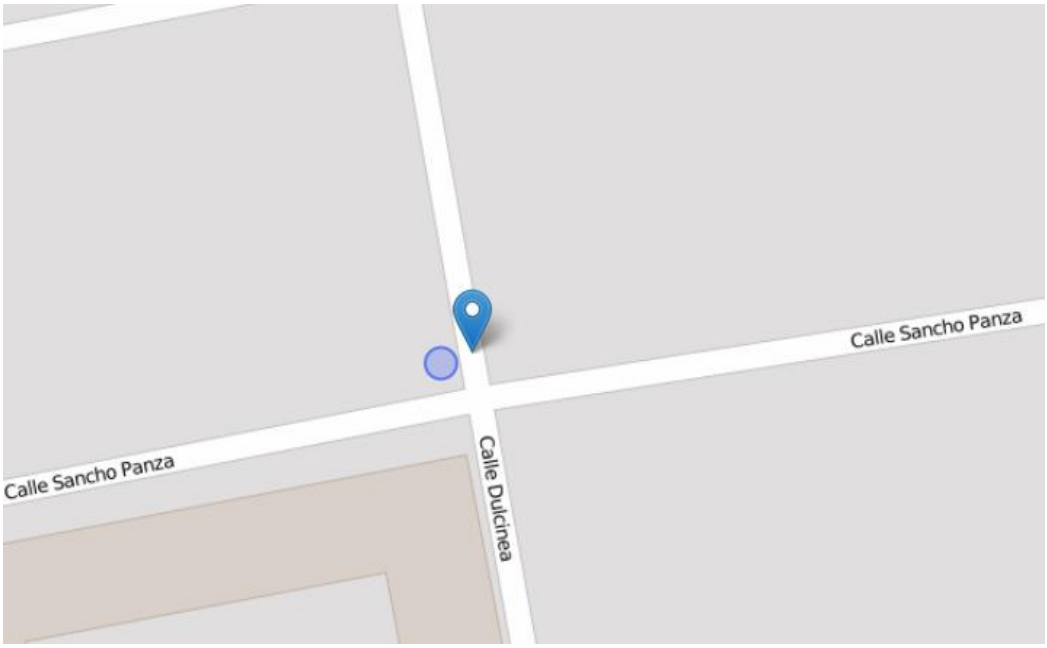

Figura 8.2 Prueba unitaria para el caso 2

Como se observa en la figura 8.2 se comprueba que se obtiene como resultado la calle más cercana a la posición del usuario en un cruce y se comprueba que este caso lo realiza correctamente.

3. Caso 3: Se encuentran dos calles cercanas, con posición anterior

El caso más común en el recorrido de un usuario al pasar por ejemplo por un cruce en un ciudad, en este caso se elige el resultado que coincida con la posición anterior, y en caso de que no se encuentre entre las opciones la calle anterior se elige la calle más cercana. Por ello se simulan ambas situaciones, obteniendo como resultados los mostrados en las figuras 8.3 y 8.4. El caso de que las dos calles más cercanas se puede dar ante una pérdida momentánea de conexión del usuario por ejemplo.

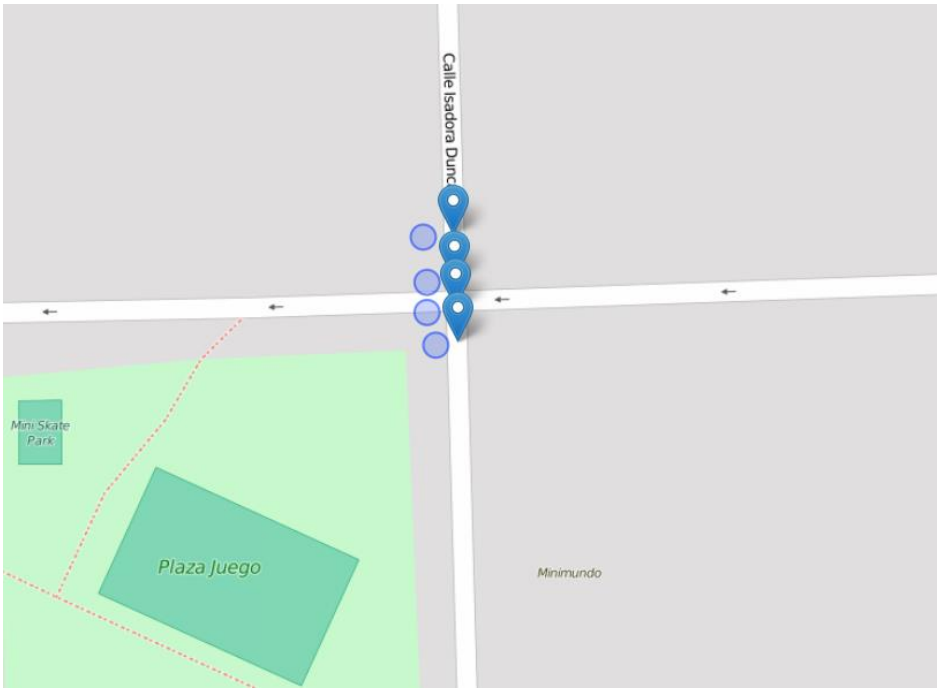

Figura 8.3 Prueba unitaria para el caso 3 donde se elige la calle anterior

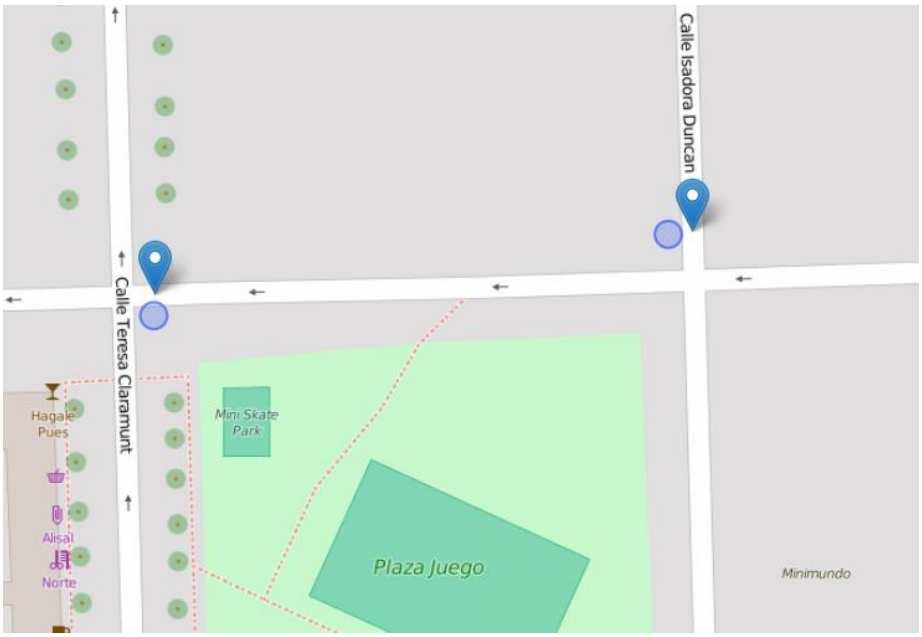

Figura 8.4 Prueba unitaria para el caso 3 donde no se elige la calle anterior

En ambas figuras se observa que los resultados son correctos y que por tanto este caso lo realiza correctamente.

Después de validar todos los casos que se pueden dar en la API, podemos dar por cerradas las pruebas unitarias ya que todas las ha pasado la aplicación con éxito. A continuación se continúa probando la API mediante pruebas de recorridos reales de un vehículo.

8.2. Pruebas de recorrido

Para poder validar el sistema completo también se realizan dos pruebas más con recorridos reales de un vehículo con el fin seguir obteniendo buenos resultados y poder validar el sistema completo. Es importante realizar pruebas con recorridos reales ya que con pruebas unitarias interactuando con el mapa es muy complejo poder determinar recorridos con la precisión real ofrecida por el GPS y las distancias entre las posiciones que ofrece el usuario.

En ambas pruebas se observan los puntos que introduce el usuario en color rojo y los resultados en color naranja. En la primera prueba se introduce al demostrador web un recorrido principalmente en autopista, por lo que se analizan situaciones en una curva y en una salida de autopista para ver el comportamiento de la API.

Primero se analiza el comportamiento en una curva tal como se indica en la figura 8.5. En ella se quiere comprobar que no se producen errores al proyectar el punto en la calle elegida y al decidir la calle.

Figura 8.5 Recorrido de un usuario en una curva

En la figura 8.5 se pueden observar dos fases, en la primera se producen múltiples cruces y se observa que la API decide bien la calle resultante. En la segunda fase se observa el comportamiento de la API en una curva donde se observa que posiciona sobre la calle los resultados correctamente.

A continuación se analiza el comportamiento en una salida de una autopista, tal como se representa en la figura 8.6, para comprobar si se cambia de calle correctamente y por tanto no se producen errores, también se analizará si se sigue realizando correctamente la proyección de los puntos en la calle.

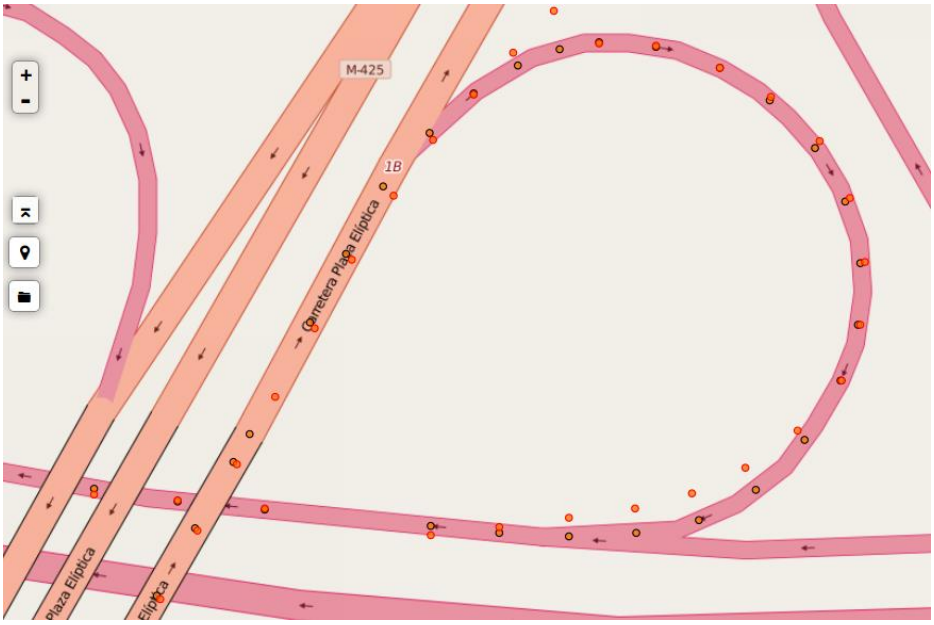

Figura 8.6 Recorrido de un usuario en una salida de autopista

En la figura 8.6 se puede observar que en una salida de una autopista también se decide correctamente la calle en la que se encuentra el usuario y también se proyectan correctamente los puntos sobre la calle decidida.

Después de validar la API para la situación anterior se prueba a continuación con un recorrido de un vehículo circulando por una ciudad para analizar casos más comunes como pueden ser intersecciones entre dos calles y rotondas.

En la primera situación de este recorrido se analiza el comportamiento en arios cruces en ciudad, como se indica en la figura 8.7. En esta situación se comprobará si la API funciona correctamente en este tipo de situaciones.

Figura 8.7 Recorrido de un usuario en un cruce

Como se observa en la figura 8.8 la API funciona correctamente para esta situación ya que proyecta los puntos sobre la calle en la que se ubica el usuario. Sin embargo en la tercera curva encontramos un error en dos puntos geográficos debido al conservadurismo de este algoritmo en la elección de la calle anterior. También se puede añadir que en el primer cruce del trayecto representado en la figura 8.8 se produjeron retenciones debido al gran número de puntos en el mismo lugar.

La siguiente situación de este recorrido a analizar se trata del recorrido del usuario en una rotonda, en esta situación se quiere comprobar si la API funciona correctamente en estas situaciones.

Figura 8.8 Recorrido de un usuario en una rotonda

En la figura 8.8 se observa que en el recorrido del usuario en una rotonda no se encuentran errores y por tanto hace correctamente la asignación y la proyección de los puntos geográficos.

Después de comprobar un correcto funcionamiento de la API en pruebas unitarias como pruebas de recorridos reales de un usuario, se realiza una prueba de rendimiento para calcular el tiempo de respuesta de la API para cada punto geográfico.

8.3. Prueba de rendimiento

Para evaluar el rendimiento de esta aplicación se realiza una prueba simple del tiempo de procesamiento de cada punto frente al tiempo de generación de los puntos por parte del usuario. Para ello se analiza una muestra de 510 puntos, del cual se calcula el tiempo total de generación por parte del usuario y se compara frente al tiempo de procesamiento en el servidor web.

En cuanto al tiempo de generación se observa que el usuario tarda 8 minutos y 50 segundos en generar 510 puntos geográficos de su recorrido. Esto implica aproximadamente 1.04 segundos por cada punto geográfico.

Sin embargo para procesar los 510 puntos geográficos en el servidor web se tarda 18 segundos. Por tanto esto significa que se tarda aproximadamente 0.035 segundos por cada punto geográfico.

Se observa que utilizando la API mediante pruebas HTTP a un servidor web local el tiempo de procesamiento es mucho menor que el tiempo de generación, lo que implica que la API funciona perfectamente en un servidor local y también se posee un amplio margen para utilizar la API mediante un servidor web que no se encuentre en local.

Después de aplicar todo este conjunto de pruebas se puede concluir que la aplicación de este trabajo funciona correctamente, con un bajo número de errores y que cumple con las especificaciones.

9. Conclusions

This section presents the conclusions after the completion of the project and possible future works for improving or expanding this project.

9.1. Conclusions

After analysing the results and make relevant evidence in this project, it can be concluded that the final application meets the specifications defined in its design.

The specifications that have been fulfilled on this project, which have been proven by tests are the following:

- Calculation of the street where the user is circulating through using the application.
- Calculation of characteristics of the street where the user is circulating, which are maximum speed and identifier of the street in the OSM database.
- Calculation of the projection on the street where the user is circulating and the direction in which circulates.
- Possibility of use of the application using HTTP requests, local application or web demonstrator.

In conclusion, this project converts GPS position data entered by users to geographic information with better quality and easy to interpret. Therefore, this work can be used to develop numerous applications.

9.2. Future works

Using as a basis this work, it can be developed applications of high interest if it has many users in a given region. This section describes three possible future work, which are explained as follows:

1. Job 1: Road safety.

With the analysis obtained via geographic data of this work using multiple users in a particular region, it can be analysed sudden changes in speed and then try to group these events to formulate hypotheses that give meaning to these sudden changes in speed and then fix them. For example, in case of sudden changes in speed obtained at the same point of a road by several users, it can distinguish two possible situations:

- It is in the OSM database a pedestrian crossing close to the point where there have been produced sudden changes in speed. This may imply that the pedestrian crossing is poorly signposted and a better signage could improve safety in that area.
- It isn't in the OSM database a crosswalk, a traffic light or an intersection where there have been sudden changes in speed. Surely in this case the street where users circulate is in a poor condition or has a bump, which difficult a proper circulation.

In conclusion, based on user geographic data that are processed in the application of this project, it can make an application that improves road safety and reduce accidents in a region.

2. Job 2: Trading analysis

This project also it can be used to improve the advertising of any company that can access to the positioning of large numbers of users. It could be used mainly in two cases:

- Analysing vehicles tours that circulate well below the maximum speed on roads, this means that they are in a jam and it can be displayed a billboard advertising in this area.
- With certain modifications in this work can also control users walking tours. With this data you can place advertising in places with a great confluence of people at certain times.

Therefore, the application of this project can serve as a basis for improving the advertising of a company that has sufficient geographic data.

3. Job 3: Traffic Analysis

This application can also provide a basis to improve city traffic. To achieve this objective, it will be analysed speeds close to a traffic light that regulates the traffic on a crossing.

With these data the possibility of changing the durations of some traffic lights, in case of that retentions are observed in only one crossing direction and not in the other direction.

Therefore, using as a basis the application of this project it can improve the city traffic, if sufficient geographic data of vehicles that circulate on the city.

In conclusion, the two studies provide an added value to a company that can be private, in the case to improve their marketing, or public, to improve the safety or traffic.

9. Conclusiones y trabajos futuros

En esta sección se expondrán las conclusiones después de realizar el trabajo y posibles mejores o trabajos futuros con las cuales ampliarlo o mejorarlo.

9.1. Conclusiones

Después de analizar los resultados y realizar las pruebas pertinentes en este trabajo se puede concluir que la aplicación final cumple con las especificaciones definidas en su diseño.

Las especificaciones que cumple este trabajo, las cuales han sido comprobadas mediante las pruebas, son las siguientes:

- Cálculo de la calle por la que circula el usuario que utiliza la aplicación.
- Cálculo de características de la calle por la que circula el usuario, que son velocidad máxima e identificador de la calle en la base de datos de OSM.
- Cálculo de la proyección sobre la calle por la que circula el usuario y el sentido en el que circula.
- Posibilidad de uso de la aplicación mediante peticiones HTTP, aplicación local o el demostrador web.

En conclusión este trabajo convierte los datos de posición GPS introducidos por usuarios en información geográfica de más calidad y más fácil de interpretar. Por tanto este trabajo se puede usar para elaborar numerosas aplicaciones.

9.2. Trabajos futuros

Empleando como base este trabajo se pueden desarrollar numerosas aplicaciones de alto interés si se dispone de numerosos usuarios en una determinada región. En esta sección se explican tres posibles trabajos futuros:

1. Trabajo 1: Seguridad vial.

Con el análisis de datos geográficos obtenidos de este trabajo a partir de múltiples usuarios, en una determinada región, se pueden analizar cambios bruscos de velocidad y después tratar de agrupar estos sucesos para obtener hipótesis que den sentido a estos cambios bruscos de velocidad y posteriormente solucionarlos. Por ejemplo en el caso de obtener cambios bruscos de velocidad en un mismo punto de una vía por parte de varios usuarios se pueden diferenciar dos posibles situaciones:

- Se encuentra en la base de datos de OSM un paso de peatones cercano al punto donde se han dado los cambios bruscos de velocidad. Esto puede implicar que el paso de peatones está mal señalizado y con una mejor señalización se podría mejorar la seguridad en esa zona.
- No se encuentra en la base de datos OSM ni un paso de peatones, semáforo o una intersección donde se han dado los cambios de velocidad. Seguramente en este caso la calle en la que circulan los usuarios se encuentre en mal estado o posea algún bache, lo que impide una correcta circulación.

En conclusión, a partir de datos geográficos de usuarios que se procesan en la aplicación de este trabajo se puede realizar una aplicación que consiga mejorar la seguridad vial y reducir accidentes de una región.

2. Trabajo 2: Análisis comercial

También se puede usar este trabajo para mejorar la publicidad de cualquier empresa que pueda acceder al posicionamiento de gran cantidad de usuarios. Se podría utilizar principalmente en dos casos:

- Analizando recorridos de vehículos que circulen muy por debajo de la velocidad máxima en carreteras, ya que esto significa que se encuentran en un atasco y pueden visualizar correctamente una valla de publicidad.
- Con ciertas modificaciones en este trabajo también se pueden controlar recorridos de usuarios a pie. Con estos datos se puede colocar publicidad en lugares con gran confluencia de personas a ciertas horas.

Por tanto la aplicación de este trabajo puede servir como base para mejorar la publicidad de una empresa que disponga de suficientes datos geográficos.

3. Trabajo 3: Análisis del tráfico

Esta aplicación también puede servir de base para mejorar el tráfico de una ciudad por ejemplo. Para conseguir este objetivo se analizarían las velocidades de los vehículos cercanos a un semáforo que regule el tránsito de vehículos en un cruce.

Con estos datos se contemplaría la posibilidad de cambiar las duraciones de algunos semáforos en el caso de que se observen retenciones en un sentido del cruce y poca afluencia de vehículos en el otro por ejemplo.

Por lo tanto, usando como base la aplicación de este trabajo se puede mejorar el tráfico de una ciudad si se poseen los suficientes datos geográficos de vehículos circulando por ésta.

En conclusión ambos trabajos proporcionarían un valor añadido a una empresa que puede ser privada, en el caso para mejorar su *marketing,* o pública, considerando la seguridad o el tráfico.

10. Marco Regulador

Para la realización de este proyecto se tienen que tener en cuenta los siguientes estándares y leyes.

Para representar puntos geográficos se emplea WGS, que es un estándar usado en cartografía, geodesia y navegación por GPS. El WGS consta de un sistema de coordenadas estándar para la Tierra, una superficie de referencia esferoidal estándar (el datum o referencia elipsoide) para los datos de altitud primas, y una superficie de potencial gravitatoria (geoide) que define el nivel del mar nominal.

La última revisión de este estándar es el WGS84, que es un sistema de referencia terrestre y datum geodésico centrado y fijo en la Tierra. WGS84 está basado en un grupo de constantes y parámetros de modelo que describen el tamaño, forma y gravedad y campos geomagnéticos de la Tierra. WGS84 [18] es la definición estándar del Departamento de Defensa de los Estados Unidos para información geoespacial y es el sistema de referencia para el GPS.

En cuanto a los datos geográficos utilizados, OSM [19] permite el uso de sus datos bajo la licencia *Open Database License* (ODbL). ODbL es un acuerdo de licencia que permite a los usuarios compartir modificar y usar la información de la base de datos mientras se mantenga esta libertad a los demás usuarios. Debido a esta licencia se eligió OSM como proveedor de datos geográficos al contrario que Google Maps que no permite modificar ni descargar sus datos.

Otro aspecto a tener en cuenta al usar recorridos reales proporcionados por usuarios es la Ley Orgánica de Protección de Datos (LOPD) [20]. Esta Ley Orgánica tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar.

En este proyecto se emplean los recorridos geográficos de los usuarios, sin asignar ni publicar ningún dato proporcionado por éstos que los relacione con sus recorridos. Por tanto se cumple con la LOPD.

En conclusión este proyecto cumple con la normativa y regulación vigente al no incumplir ninguna norma y no necesitar ninguna licencia de uso de los datos geográficos provenientes de OSM.

11. Entorno socioeconómico

Cada vez más aplicaciones utilizan la geolocalización del usuario para desarrollar aplicaciones de mejor calidad. El hecho de conocer la posición del usuario permite mejorar ciertos aspectos en la aplicación, por ejemplo:

- Acceder a la geolocalización del usuario puede permitir conocer al desarrollador variedad de información del cliente como son las zonas por la que circula y sus horarios. Esta información resulta muy útil para diversas campañas de *marketing*. Siempre y cuando el usuario acepte las clausulas de privacidad correspondientes.
- Permite hacer *marketing* de proximidad. Por ejemplo si el desarrollador de la aplicación se tratara de un restaurante. Éste podría notificar al usuario con notificaciones *push* sobre las ofertas del día cuando el usuario se encuentre situado en la misma calle en la que se encuentra su local.
- Además permite una mayor cercanía con el usuario y por tanto mejora la eficiencia de la aplicación.

Por tanto queda demostrado que conocer la geolocalización del usuario, en concreto la vía por la que circula, puede mejorar notablemente la experiencia del usuario y por tanto reportar más beneficios para el desarrollador.

Existen muchos servicios en línea que pueden ofrecer la calle por la que circula el usuario a partir de la posición GPS del usuario como son Google Maps y OSM. El primero no es totalmente libre y se necesita de una conexión a Internet. El segundo también necesita de una conexión a Internet y haría al desarrollador de la aplicación depender de los servidores de OSM para su funcionamiento correcto.

Por estas razones proporcionar a los desarrolladores una biblioteca de código libre que puedan integrar en sus aplicaciones facilitaría la proliferación de las mismas, lo cual a su vez supondría un beneficio para la sociedad en su conjunto.

12. Bibliografía

[1] F. H. S. S. C. V. A. Christine Najar, «What is GIS?,» [En línea]. Available: http://www.gitta.info/what_gis/en/text/what_gis.pdf.

[2] «Instituto Geográfico Nacional de la República Argentina,» [En línea]. Available: http://www.ign.gob.ar/NuestrasActividades/ProduccionCartografica/Introduccion.

[3] I. A. Fernández-Coppel, «El Datum,» [En línea]. Available: http://www.cartesia.org/data/apuntes/cartografia/cartografia-datum.pdf. [Último acceso: 12 07 2016].

[4] BOE, «Boletin Oficial del Estado,» 29 08 2007. [En línea]. Available: http://www.boe.es/boe/dias/2007/08/29/pdfs/A35986-35989.pdf. [Último acceso: 12 07 2016].

[5] A. El-Rabbany, Introduction to GPS: The Global Positioning System.

[6] Instituto Geográfico Nacional, «Cartografía y bases geográficas,» [En línea]. Available: https://www.ign.es/ign/layoutIn/actividadesBDGintro.do.

[7] OpenStreetMap, «Wiki OpenStreetMap,» [En línea]. Available: https://wiki.openstreetmap.org/wiki.

[8] OpenStreetMap, «Sitio web de OSM,» [En línea]. Available: http://www.openstreetmap.org.

[9] Leaflet, «Sitio web de Leaflet,» [En línea]. Available: http://leafletjs.com/.

[10] PostgreSQL, «Sitio web PostgreSQL,» [En línea]. Available: https://www.postgresql.org/.

[11] PostGIS, «Sitio web de PostGIS,» [En línea]. Available: http://postgis.net/.

[12] Google , «Detalles sobre la API de Google Maps,» [En línea]. Available: https://developers.google.com/maps/pricing-and-plans/#details.

[13] Postgis, «Documentación de Postgis,» [En línea]. Available: http://postgis.net/stuff/postgis-2.2.pdf. [Último acceso: 03 08 2016].

[14] «Documentación de leaflets,» [En línea]. Available: http://leafletjs.com/reference.html.

[15] «API JQuery,» [En línea]. Available: http://api.jquery.com/.

[16] «API de EasyButton,» [En línea]. Available: https://github.com/CliffCloud/Leaflet.EasyButton.

[17] «IonIcons,» [En línea]. Available: http://ionicons.com/.

[18] «WGS84,» [En línea]. Available: https://confluence.qps.nl/pages/viewpage.action?pageId=42315395#WorldGeodeticSystem1984(WGS84)-DefinicionesWGS84.

[19] OpenStreetMaps, «Wiki Legal de OSM,» [En línea]. Available: http://wiki.openstreetmap.org/wiki/ES:Legal_FAQ.

[20] «LOPD,» [En línea]. Available: http://noticias.juridicas.com/base_datos/Admin/lo15-1999.t1.html#t1.